```
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRRRRRRRRRRR           III       VVV         VVV  EEEEEEEEEEE      RRRRRRRRRRRR
DDD          DDD RRRRRRRRRRRR           III       VVV         VVV  EEEEEEEEEEE      RRRRRRRRRRRR
DDD          DDD RRRRRRRRRRRR           III       VVV         VVV  EEEEEEEEEEE      RRRRRRRRRRRR
DDD          DDD RRR   RRR              III       VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR   RRR              III       VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR   RRR              III       VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR      RRR           III          VVV   VVV     EEE              RRR      RRR
DDD          DDD RRR      RRR           III          VVV   VVV     EEE              RRR      RRR
DDD          DDD RRR      RRR           III          VVV   VVV     EEE              RRR      RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII          VVV       EEEEEEEEEEEEEEE  RRR         RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII          VVV       EEEEEEEEEEEEEEE  RRR         RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII          VVV       EEEEEEEEEEEEEEE  RRR         RRR
```

```
XX      XX  EEEEEEEEEE  DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX  EEEEEEEEEE  DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX  EE          DD      DD  RR      RR      II      VV      VV  EE          RR      RR
XX      XX  EE          DD      DD  RR      RR      II      VV      VV  EE          RR      RR
  XX  XX    EE          DD      DD  RR      RR      II      VV      VV  EE          RR      RR
  XX  XX    EE          DD      DD  RR      RR      II      VV      VV  EE          RR      RR
    XX      EEEEEEEE    DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
    XX      EEEEEEEE    DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
  XX  XX    EE          DD      DD  RR  RR          II      VV      VV  EE          RR  RR
  XX  XX    EE          DD      DD  RR  RR          II      VV      VV  EE          RR  RR
XX      XX  EE          DD      DD  RR    RR        II        VV  VV    EE          RR    RR        ....
XX      XX  EE          DD      DD  RR    RR        II        VV  VV    EE          RR    RR        ....
XX      XX  EEEEEEEEEE  DDDDDDDD    RR      RR    IIIIII        VV      EEEEEEEEEE  RR      RR      ....
XX      XX  EEEEEEEEEE  DDDDDDDD    RR      RR    IIIIII        VV      EEEEEEEEEE  RR      RR      ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

(11) 1573 DEUNA/DELUA device driver diagnostic function codes
(12) 1586 Standard tables
(13) 1673 Local driver storage
(14) 1948 CONTROL_INIT - INITIALIZE DEUNA/DELUA DEVICE
(15) 1974 CLONED_UCB - INITIALIZE THE CLONED UCB
(16) 2004 UNIT_INIT - INITIALIZE THE DEUNA/DELUA UNIT
(17) 2132 FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
(18) 2171 XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
(19) 2279 XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
(20) 2343 XMT_START - START TRANSMIT OPERATION
(21) 2612 RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
(22) 2692 RCV_START - START RECEIVE I/O OPERATION
(23) 2772 SUBROUTINES TO FIND SHR DATA STRUCTURE
(24) 2834 ALT_START - ALTERNATE START I/O ROUTINE
(25) 2904 SETMODE_FDT - SET MODE I/O OPERATION FDT DISPATCH ROUTINE
(27) 3329 SHR_UCB - CREATE SHARED UCB
(28) 3469 CHECK_PARAM - CHECK SHARED USERS PARAMETERS
(29) 3531 SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
(30) 3563 SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
(31) 3698 READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
(32) 3788 DIAGNOSE_FDT - DIAGNOSTIC OPERATION FDT DISPATCH ROUTINE
(33) 3981 GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
(34) 4037 CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
(35) 4087 CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
(36) 4126 ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
(37) 4193 CHECK_DGBUFx - CHECK ACCESS TO DIAGNOSTIC BUFFER
(38) 4265 STARTIO - START I/O OPERATION
(39) 4354 START - START UNIT'S PROTOCOL
(40) 4813 INPUT - INSERT REQUEST ON INPUT PROCESS QUEUE
(41) 4844 INPUT_BUFFER - INSERT REQUEST ON INPUT PROCESS QUEUE AND MAP BUFFER
(42) 4927 FILLRCVLIST - FILL RECEIVE BUFFER LIST
(43) 5009 START_RECEIVE - START ANY RECEIVE REQUESTS PENDING
(44) 5127 LOAD_PORT - LOAD CSR'S WITH COMMAND REQUEST
(45) 5508 DONE_INTR - DEUNA/DELUA COMPLETION INTERRUPT SERVICE ROUTINE
(46) 5621 SCHED_FORK - SCHEDULE THE FORK PROCESS
(46) 5622 SCHED_FORKC - SCHEDULE THE FORK PROCESS WITH R3 CLEAR
(47) 5666 FORK_PROC - Error and completion fork process handling
(48) 5934 SCAN_LISTS - Scan transmit and receive ring entries
(49) 6193 SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
(50) 6255 COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
(51) 6322 FINISH_XMT_FFI - Finish FAST interface transmit processing
(52) 6354 FINISH_RCV_FFI - Finish FAST receive processing
(53) 6400 FINISH_RCV_IO - Finish receive I/O processing
(54) 6553 ASSPKT_NODCH - Assemble receive packets without Data Chaining
(55) 6607 ASSPKT_DCH - Assemble receive packets with Data Chaining Enabled
(56) 6811 MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
(57) 6860 MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
(58) 6932 REG_DUMP - DEUNA/DELUA ERROR LOG AND DIAGNOSTICS REGISTER DUMP
(59) 6984 RES.ART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
(60) 7074 TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
(61) 7155 TIMEOUT - TIMEOUT SERVICE ROUTINE
(62) 7205 ALLOC_CDB - ALLOCATE THE CDB
(63) 7272 SHUTDOWN_UNA - SHUTDOWN DEUNA/DELUA AND ALL UNITS
(64) 7423 IRP_ABORTED - Abort an IRP on one of the CDB queues
(65) 7493 SHUTDOWN - SHUT DOWN UNIT
(65) 7494 SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
(66) 7759 BLD_STOP_IRP - Build an IRP to reset promiscuous mode
(67) 7798 BLD_STRT_IRP - Build a point-to-point startup IRP
(68) 7843 BLD_IRP - Build an IRP routine
(68) 7844 BLD_IRP_CO - Build an IRP co-routine

```
(69)    7916        CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE                                   XE
(69)    7917        DELETE_SHR - DELETE SHR DATA STRUCTURE                                                  VC
(70)    8033        CANCEL - CANCEL I/O ON UNIT
(71)    8225        SUBROUTINES TO FIND SHR DATA STRUCTURE GIVEN PCB AND CHAN
(72)    8282        FIND_POINT_UCB  - Find the point to point UCB
(73)    8371        ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
(74)    8442        MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
(75)    8472        ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
(76)    8508        VALIDATE_P2 - VALIDATE P2 BUFFER PARAMETERS
(77)    8690        CHANGE_PARAM - UPDATE UCB/CDB BASED ON P2 BUFFER PARAMETERS
(78)    8818        RETURN_P2,  Return UCB/CDB buffer parameters
(79)    8925        VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
(80)    8982        VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
(81)    9044        SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
(82)    9198        SET_PHYAD - SET THE PHYSICAL ADDRESS
(82)    9199        SET_DESAD - SET THE DESTINATION ADDRESS
(83)    9281        RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
(84)    9324        MATCH_MULTI - CHECK MULTICAST ADDRESS
(85)    9369        MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
(86)    9411        POKE_USER - DELIVER ATTENTION ASTS
(87)    9453        MATCH_PROTYP - Match protocol type
(87)    9454        MATCH_PROMTYP - Find the promiscuous user
```

```
0000     1          .TITLE  XEDRIVER - VAX/VMS DEUNA/DELUA Driver
0000     2          .IDENT  'V04-000'
0000     3 ;
0000     4 ;*****************************************************************************
0000     5 ;*                                                                           *
0000     6 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8 ;*  ALL RIGHTS RESERVED.                                                     *
0000     9 ;*                                                                           *
0000    10 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    11 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    12 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    13 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    14 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    15 ;*  TRANSFERRED.                                                             *
0000    16 ;*                                                                           *
0000    17 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    18 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    19 ;*  CORPORATION.                                                             *
0000    20 ;*                                                                           *
0000    21 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    22 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23 ;*                                                                           *
0000    24 ;*                                                                           *
0000    25 ;*****************************************************************************
0000    26 ;++
0000    27 ; FACILITY:
0000    28 ;
0000    29 ;       VAX/VMS DEUNA/DELUA QUEUE I/O DRIVER
0000    30 ;
0000    31 ; ABSTRACT:
0000    32 ;
0000    33 ;       This module contains the DEUNA/DELUA driver FDT routines,
0000    34 ;       interrupt dispatcher, interrupt service and fork routines.
0000    35 ;
0000    36 ; AUTHOR:
0000    37 ;
0000    38 ;       Rod Gamache     21-Oct-1981
0000    39 ;
0000    40 ;
0000    41 ; MODIFICATION HISTORY:
0000    42 ;
0000    43 ;       V03-034 RNG0034         Rod Gamache             30-Aug-1984
0000    44 ;               Do not mark template UCB OFFLINE, simply because the
0000    45 ;               device reports a hardware error, that may be cleared up.
0000    46 ;
0000    47 ;       V03-033 RNG0033         Rod Gamache             2-Aug-1984
0000    48 ;               Fix check for ready state on DELUA. Also increase done
0000    49 ;               completion interrupt expected time to 16 seconds.
0000    50 ;
0000    51 ;       V03-032 RNG0032         Rod Gamache             25-Jul-1984
0000    52 ;               Fix powerfail recovery code to clear the POWERFAIL indicator
0000    53 ;               bit and set the ONLINE bit.
0000    54 ;
0000    55 ;       V03-031 LMP0275         L. Mark Pilant,         12-Jul-1984  12:43
0000    56 ;               Initialize the ACL info in the ORB to be a null descriptor
0000    57 ;               list rather than an empty queue.  This avoids the overhead
```

```
0000     58 ;                of locking and unlocking the ACL mutex, only to find out
0000     59 ;                that the ACL was empty.
0000     60 ;
0000     61 ;        V03-030 RNG0030         Rod Gamache             27-Jun-1984
0000     62 ;                Change the way the set default modifier behaves on the
0000     63 ;                Physical address parameter.
0000     64 ;
0000     65 ;        V03-029 LMP0221         L. Mark Pilant,         27-Mar-1984  11:55
0000     66 ;                Change UCB$L_OWNUIC to ORB$L_OWNER and UCB$W_VPROT to
0000     67 ;                ORB$W_PROT.
0000     68 ;
0000     69 ;        V03-028 RNG0028         Rod N. Gamache          29-Feb-1984
0000     70 ;                Fix return from EXE$PROBE_DSC to setup R1 as a longword.
0000     71 ;                Fix ALLOC_CDB to initialize the address of UCB unit 0.
0000     72 ;                Fix error path to return transmit CXBs on FFI interface,
0000     73 ;                instead of deallocating them.
0000     74 ;
0000     75 ;        V03-027 TMK0007         Todd M. Katz            06-Feb-1984
0000     76 ;                Fix a bug in ALLOC_CDB. When scanning the list of UCBs within
0000     77 ;                ALLOC_CDB to setup UCB$L_NI_HWAPTR and UCB$L_CDB, make sure that
0000     78 ;                the scan is complete and that these locations are setup in
0000     79 ;                every UCB, and not just the UCBs from UCB0 to the current UCB
0000     80 ;                (the UCB the routine was called with) inclusive.
0000     81 ;
0000     82 ;        V03-026 RNG0026         Rod N. Gamache          03-Feb-1984
0000     83 ;                Add a delay to the startup sequence to give the DEUNA time
0000     84 ;                to set the hardware state to RUN. This will allow the driver
0000     85 ;                the oportunity to return this state to UETP. Also the driver
0000     86 ;                will have to clear BIT13 in PCSR1 for UETP so that VAX 11/725s
0000     87 ;                will pass UETP. Also fix problem where correct error return
0000     88 ;                was not getting returned when quota was exceeded.
0000     89 ;
0000     90 ;        V03-025 TMK0006         Todd M. Katz            01-Feb-1984
0000     91 ;                When the DEUNA/DELUA times out because of UNIBUS or device
0000     92 ;                controller power failure, then call back all protocols which
0000     93 ;                are using the FFI interface and have defined an FFI$L_ERROR
0000     94 ;                asynchronous error routine. What this involves is:
0000     95 ;
0000     96 ;                1. Not clearing the power failure bit, CDB_STS_V_POWER, within
0000     97 ;                   CDB_B_STS in the routine SHUTDOWN_UNA until all I/O has been
0000     98 ;                   cleaned up on all active units.
0000     99 ;
0000    100 ;                2. Noticing within the routine SHUTDOWN by means of the
0000    101 ;                   CDB_STS_V_POWER status bit that a power failure has
0000    102 ;                   occurred. In such a circumstance the port driver wants to
0000    103 ;                   call the asynchronous error routine of the protocol before
0000    104 ;                   doing anything else provided the protocol has initialized
0000    105 ;                   the FFI interface, defined an asynchronous error routine,
0000    106 ;                   and the UCB for the protocol is both on-line and initialized.
0000    107 ;
0000    108 ;        V03-024 TMK0005         Todd M. Katz            19-Jan-1984
0000    109 ;                Make the following changes to the driver:
0000    110 ;
0000    111 ;                1. I have created a NI device dependent UCB extension within
0000    112 ;                   $UCBDEF. This extension contains definitions for
0000    113 ;                   UCB$L_NI_HWAPTR and UCB$L_NI_MLTPTR, two new locations
0000    114 ;                   to be contained within the UCBs of all NI datalink drivers.
```

```
0000   115 ;                I have therefore modified the XEDRIVER's UCB definition
0000   116 ;                so that the DEUNA/DELUA specific UCB fields begin immediately
0000   117 ;                following the NI device dependent UCB extension.
0000   118 ;
0000   119 ;                UCB$L_NI_HWAPTR is initialized when the CDB is first
0000   120 ;                allocated to contain the address of CDB_G_HWA, the CDB
0000   121 ;                location which contains the NI device's unique hardware
0000   122 ;                address. UCB$L_NI_MLTPTR is initialized within the unit
0000   123 ;                unitialization routine to contain the address of the table
0000   124 ;                of multicast addresses defined for this protocol type.
0000   125 ;                Both of these values must be accessible to the NI-SCS port
0000   126 ;                driver. The NI-SCS port driver has access to the UCBs of the
0000   127 ;                NI devices participating in SCS clusters, but it doesn't have
0000   128 ;                initimate knowledge of how the UCBs and CDBs are layed out
0000   129 ;                for each NI device. This UCB extension provides a means for
0000   130 ;                the NI-SCS port driver to locate these values without knowing
0000   131 ;                the exact layout of each of the NI device's UCB and CDB.
0000   132 ;
0000   133 ;             2. Whenever a protocol is to be started up on a DEUNA/DELUA,
0000   134 ;                allow the initiator of the SETMODE+STARTUP to specify 0
0000   135 ;                receive buffers instead of the former minimum of 1. The
0000   136 ;                effect on this protocol is that it must have a READ
0000   137 ;                outstanding at all times in order to guarentee that it will
0000   138 ;                receive all datagrams specifying its protocol. If a datagram
0000   139 ;                was received for this protocol, and there wasn't a read
0000   140 ;                outstanding, then because 0 receive buffers can be queued
0000   141 ;                (or saved) for this protocol, the receive buffer would be
0000   142 ;                deallocated to pool, and the message it contained lost.
0000   143 ;
0000   144 ;                This change is extremely useful for those users who are
0000   145 ;                making use of the FFI interface provided by this port driver.
0000   146 ;                Between the time the user issues a SETMODE+STARTUP, and the
0000   147 ;                time the user initializes the FFI interface by calling
0000   148 ;                FFI_INIT, it would be possible for the port driver to have
0000   149 ;                received messages with this user's protocol, and to have
0000   150 ;                queued them up to the appropriate UCB. Now, if the user
0000   151 ;                never makes use of the QIO or ALTSTART interfaces, but just
0000   152 ;                uses the FFI interfaces for communication, these messages,
0000   153 ;                which are probably stale by this time, will never be
0000   154 ;                received, and the buffers that contain them will effectively
0000   155 ;                be lost forever. With this change, a user who wishes to do
0000   156 ;                all his/her communication via the FFI interface can guarentee
0000   157 ;                that a situation such as this can never arise, and buffers
0000   158 ;                such as these can never be "lost".
0000   159 ;
0000   160 ;             4. Add the capability of requesting that the physical address
0000   161 ;                of the DEUNA/DELUA device be set to the default DECnet
0000   162 ;                address, when the device is first initialized. This new
0000   163 ;                capability is requested by means of the NMA$C_LNMCN_SDF
0000   164 ;                mode value specified within the modifier field of the
0000   165 ;                NMA$C_PCLI_PHA parameter.
0000   166 ;
0000   167 ;        V03-023 TMK0004         Todd M. Katz            17-Dec-1983
0000   168 ;                Fix a bug introduced by TMK0003. When the controller is a DELUA
0000   169 ;                I was re-setting UCB$B_TYPE instead of UCB$B_DEVTYPE to
0000   170 ;                DT$_DELUA.
0000   171 ;
```

```
0000   172 ;               I have also added code to UNIT_INIT which during the
0000   173 ;               initialization of UCB0 determines the state of the DEUNA/DELUA.
0000   174 ;               If the device is in a state other than PRIMARY LOAD, READY, or
0000   175 ;               RUNNING, I now clear the on-line bit within the UCB, and skip
0000   176 ;               the determination of the device's identification because the
0000   177 ;               device ID field within PSCR1 is probably invalid.
0000   178 ;
0000   179 ;       V03-022 TMK0003         Todd M. Katz            10-Nov-1983
0000   180 ;               Add support for the DELUA. Support consists of the following
0000   181 ;               modifications to the DEUNA driver:
0000   182 ;
0000   183 ;               1. The addition of an Extensive Selftest Port Command. This
0000   184 ;                  command maybe invoked as a diagnositic function
0000   185 ;                  (XE_C_DG_ESTEST) on the DELUA only.
0000   186 ;
0000   187 ;               2. The addition of a HALT Port Command. This command maybe
0000   188 ;                  invoked as a diagnositic function (XE$C_DG_HALT) on the
0000   189 ;                  DELUA only. Also, add a definition for the the new port
0000   190 ;                  state, PORT HALTED, that only the DELUA can enter.
0000   191 ;
0000   192 ;               3. The addition of an internal loopback mode for the DELUA
0000   193 ;                  only. Internal loopback maybe requested by specifying the
0000   194 ;                  controller mode (Line parameter NMA$C_PCLI_CON) as
0000   195 ;                  NMA$C_LINCN_LOO and by specifying the new line parameter
0000   196 ;                  NMA$C_PCLI_ILP with a value of NMA$C_STATE_ON. Requesting
0000   197 ;                  internal loopback results in the setting of the new port
0000   198 ;                  control block mode definition bit INTL.
0000   199 ;
0000   200 ;               4. The addition of a new interrupt bit, FATL (Fatal Error
0000   201 ;                  Interrupt), as bit <09> in PCSR0. When a controller interrupt
0000   202 ;                  occurs, and the port driver notices that this bit has been
0000   203 ;                  set then the contents of PCSR1 are ignored during processing
0000   204 ;                  of the error by FORK_PROC. This is accomplished by zeroing
0000   205 ;                  the high order word of R3 before calling SCHED_FORK to
0000   206 ;                  schedule a fork process to handle the error. The fork
0000   207 ;                  process assumes, when it starts executing, that the high
0000   208 ;                  order word of R3 contains the contents of PCSR1.
0000   209 ;
0000   210 ;               5. Add to PCSR1 a three bit field DEVID (Device Identification)
0000   211 ;                  from bit <04> to bit <06>, and definitions for several
0000   212 ;                  possible bit settings. This device identification field
0000   213 ;                  is used to identify a DELUA from a DEUNA. At unit
0000   214 ;                  initialization time for UCB0, the template UCB, this field
0000   215 ;                  is queried, and if the controller is a DELUA, the device
0000   216 ;                  type field of the UCB is changed from its default of
0000   217 ;                  DT$_DEUNA to DT$_DELUA. This field is then subsequently
0000   218 ;                  referenced whenever the port driver must distinguish between
0000   219 ;                  the two types of controllers. The addition of this new field
0000   220 ;                  requires removal of the bit definitions for POWER (bit <04>
0000   221 ;                  Software Defined Power Fail) and for TIMOUT (bit <05>
0000   222 ;                  Software Defined Timeout) as well as all code in the driver
0000   223 ;                  that sets and references these bits. The latter bit is
0000   224 ;                  redundant (and thus not needed) and the function of the
0000   225 ;                  former is taken up be a new CDB status bit (described below).
0000   226 ;
0000   227 ;               6. The transmit ring descriptor entries for the DELUA define a
0000   228 ;                  new error bit, UFLO (bit <13> - Transmit Underflow). When
```

```
0000   229 ;                this bit is detected as being set in a completed transmit,
0000   230 ;                the transmit is concluded with an error status of
0000   231 ;                SS$_CNTRLERR - Fatal Controller Error.
0000   232 ;
0000   233 ;             7. The recieve ring descriptor entries for the DELUA define a
0000   234 ;                new error bit, OVRN (bit <12> - Recieve Overrun). This
0000   235 ;                error status is not treated any differently from any of the
0000   236 ;                other recieve error statuses - the recieve buffer(s) are
0000   237 ;                returned to the free list (or to pool if the free list is
0000   238 ;                filled), and the scan for recieved messages continues.
0000   239 ;
0000   240 ;             8. The DELUA always operates in half-duplex mode (except for
0000   241 ;                loopback); therefore, the port driver has been modified to
0000   242 ;                not allow the echo-back line parameter (NMA$C_PCLI_EKC) to
0000   243 ;                be specified when the controller is a DELUA, and to always
0000   244 ;                made sure that the port control block mode half-duplex
0000   245 ;                definition bit (HDX) is clear.
0000   246 ;
0000   247 ;             Other changes made to support the DELUA and to clean up the
0000   248 ;             existing driver are as follows:
0000   249 ;
0000   250 ;             1. Change the name of bit <08> of PCSR0 from FATI to USCI
0000   251 ;                (Unsolicitated State Change Interrupt).
0000   252 ;
0000   253 ;             2. Move the software defined illegal interrupt bit, ILLINT,
0000   254 ;                from bit <06> of PCSR1 to bit <04> of PCSR0.
0000   255 ;
0000   256 ;             3. Eliminate the port commands TMRO (Sanity Timer On), TMRF
0000   257 ;                (Sanity Timer Off), and RSTT (Reset Sanity Timer).
0000   258 ;
0000   259 ;             4. Increase the size of the STATE field of PCSR1 from 3 to 4
0000   260 ;                bits. STATE now occupies from bit <00> to bit <03> of PCSR1.
0000   261 ;                This change requires the removal of the bit definition for
0000   262 ;                RMTC (Remote Console Reserved) bit <03> from PCSR1.
0000   263 ;
0000   264 ;             5. Fix power failure recovery which has not been working for
0000   265 ;                some time. Power failure recovery has not been working
0000   266 ;                because the CDB address was not being loaded into R4 before
0000   267 ;                SCHED_FORK was being invoked from within TIMOUT to setup
0000   268 ;                a fork process to handle the error. Since the fork process
0000   269 ;                assumes that on entry R4 contains the CDB address, and R4
0000   270 ;                did not contain the CDB address in the case of power failures
0000   271 ;                this explains the problems which have been seen. In addition,
0000   272 ;                a CDB status bit, CDB_STS_V_POWER, has been defined. This bit
0000   273 ;                is now set (for reference purposes only) whenever a power
0000   274 ;                failure occurrs.
0000   275 ;
0000   276 ;             6. Add comments to explicitely distinguish DEUNA parameters
0000   277 ;                and operations from DELUA parameters and operations, and
0000   278 ;                visa versa.
0000   279 ;
0000   280 ;             7. Change the DEUNA device type from DT$_UNA11 to DT$_DEUNA.
0000   281 ;
0000   282 ;             8. When the CDB address is retrieved for the first time during a
0000   283 ;                transmit, terminate the operation with the appropriate error
0000   284 ;                if there is no CDB.
0000   285 ;
```

```
0000    286 ;               9. When COPY_RCV determines that the promiscious user does not
0000    287 ;                  have a waiting IRP and that the recieve can not be buffered
0000    288 ;                  because of insufficient quota it returns the buffer to the
0000    289 ;                  pool. It makes the assumption when it goes to return such a
0000    290 ;                  buffer, that the buffer can be chained, even though such
0000    291 ;                  chained buffers are not supported. Therefore, this routine
0000    292 ;                  goes through some unnecessary gymnastics to deallocate such
0000    293 ;                  buffers, and this change cleans up this buffer deallocation
0000    294 ;                  procedure.
0000    295 ;
0000    296 ;              10. There currently is a bug in $DASSGNs on shared UCBs
0000    297 ;                  (ie - the protocol is being shared) when the UCB is on-line
0000    298 ;                  and the reference count has not gone to zero. In this case
0000    299 ;                  a selective cancel I/O is done by the routine CANCEL. The
0000    300 ;                  problem is that for this selective cancel to be done, R4
0000    301 ;                  must contain the PCB address. Unfortunately it is possible
0000    302 ;                  for R4 to be over-written. Therefore, the fix is to make sure
0000    303 ;                  that R4 contains the PCB address, when it is expected to
0000    304 ;                  contain the PCB address.
0000    305 ;
0000    306 ;       V03-021 TMK0002         Todd M. Katz            10-Nov-1983
0000    307 ;               Performance enahcements to the transmit and recieve main code
0000    308 ;               paths are as follows:
0000    309 ;
0000    310 ;                  1. I eliminated the check for runt packets that was being
0000    311 ;                     made within XMT_START if padding is not enabled. The
0000    312 ;                     assumption will be made that since padding is enabled by
0000    313 ;                     default, if the user turns it off then he/she should know
0000    314 ;                     enough to not attempt to transmit runt packets. In addition
0000    315 ;                     there was one place in XMT_START that was setting up the
0000    316 ;                     address of the CDB in R0 even though it already had the CDB
0000    317 ;                     address in R4, and I removed the unnecessary instructions.
0000    318 ;
0000    319 ;                  2. I modified ADDRCVLIST as follows. When this routine is called
0000    320 ;                     R2 contains the address of a CXB that is to be added to the
0000    321 ;                     recieve buffer list if there is sufficient quota to do so.
0000    322 ;                     I have made the assumption, that the CXB$B_TYPE and
0000    323 ;                     CXB$W_SIZE fields of the CXB have been appropriately
0000    324 ;                     initialized, and there is no need to do so again. This
0000    325 ;                     serves to eliminate four instructions in the case when there
0000    326 ;                     is sufficient quota to add the recieve buffer to the list,
0000    327 ;                     and one instruction from the path when the port driver must
0000    328 ;                     dip into the additional quota in order to add the recieve
0000    329 ;                     buffer to the list.
0000    330 ;
0000    331 ;                  3. START_RECIEVE, as part of its job of mapping recieve buffers
0000    332 ;                     and handing them to the device, was computing the size of
0000    333 ;                     each recieve buffer and stuffing it in to the ring
0000    334 ;                     descriptor that it was currently initiating. This is
0000    335 ;                     unnecessary because the size of all recieve buffers is fixed
0000    336 ;                     at the time the CDB is first initiated and the controller is
0000    337 ;                     started. Therefore, what I have done is to initialize the
0000    338 ;                     buffer length fields of all ring descriptor entries as part
0000    339 ;                     of STARTUP, and remove the individual recieve descriptor
0000    340 ;                     buffer length initializations within START_RECIEVE.
0000    341 ;
0000    342 ;                  4. I have made a modification to LOAD_PORT which should result
```

```
0000   343 ;
0000   344 ;
0000   345 ;
0000   346 ;
0000   347 ;
0000   348 ;
0000   349 ;
0000   350 ;
0000   351 ;
0000   352 ;
0000   353 ;
0000   354 ;
0000   355 ;
0000   356 ;
0000   357 ;
0000   358 ;
0000   359 ;
0000   360 ;
0000   361 ;
0000   362 ;
0000   363 ;
0000   364 ;
0000   365 ;
0000   366 ;
0000   367 ;
0000   368 ;
0000   369 ;
0000   370 ;
0000   371 ;
0000   372 ;
0000   373 ;
0000   374 ;
0000   375 ;
0000   376 ;
0000   377 ;
0000   378 ;
0000   379 ;
0000   380 ;
0000   381 ;
0000   382 ;
0000   383 ;
0000   384 ;
0000   385 ;
0000   386 ;
0000   387 ;
0000   388 ;
0000   389 ;
0000   390 ;
0000   391 ;
0000   392 ;
0000   393 ;
0000   394 ;
0000   395 ;
0000   396 ;
0000   397 ;
0000   398 ;
0000   399 ;
```

in increased performance. Currently when this routine is called the assumption is made that the port will either be busy or there will be nothing to load it with. Consequently, the state bit indicating the status of the port, CDB_STS_V_FUNC, is not cleared (indicating that the port is active) unless the port was found to be free and there is a command to load it with. I think a more appropriate assumption is that when this routine is called, regardless of the state of the port, there will be command to load it with if the port is free. Therefore, what I have done is to always clear CDB_STS_V_FUNC while checking it by means of a BBCC. If the port is busy, then the clearing of this bit has no effect because it is already 0. If the port is free and there is a command to load it with, then the BICW currently being done in this case becomes unnecessary, and can be removed. Finally, if the port is free but there is no command to load, which should be the least frequent case, then CDB_STS_V_FUNC must be set before this routine exits and I have added a BISW to do this.

A second improvement is to not obtain the PCBB address until after it is determined that the next function is not a CXB function but an IRP function. This eliminates an unnecessary instructions from the main transmit and recieve code paths.

6. There are two consecutive MOVL within XMT_START that can be replaced with a single MOVQ.

7. One of the first things XMT_ALT_START does is to retrieve the CDB address even though it already has the address of the CDB within R4. Eliminate the unnecessary MOVL.

8. The routine FINISH_RCV_FFI currently saves the CDB address in R1 before loading the FFI block address into R4 where the CDB address is normally kept. By delaying the retrieving of the FFI block address until just before the user is called back with the CXB, the MOVL that saves the CDB address in R1 can be eliminated.

9. At the present time, the address of the CDB is retrieved via a chain of pointers within the following data structures:

   UCB -> CRB -> CDB

   By defining a location within the device-dependant portion of the UCB for the CDB address, and by setting up this location initially when the CDB is first allocated, a MOVL instruction can be eliminated for each time the CDB address must be retrieved.

10. Finally, the port driver assumes data chaining by default, and this mandates a complicated song and dance to construct recieve packets despite the fact that DECNET specifies the maximum recieve buffer possible and data chaining can never take place! I have changed this my turning off data chaining as the default, not allowing the controller to be enabled for data chaining unless the initializer of the controller has

```
0000    400 ;                   explicitely specified data chaining and a buffer size less
0000    401 ;                   than the maximum packet size, and setting up 2 assemble
0000    402 ;                   packet routines - one that supports data chaining,
0000    403 ;                   ASSPKT_DCH, and one that does not ASSPKT_NODCH. When the
0000    404 ;                   controller is initialized as part of a STARTUP, a decision
0000    405 ;                   is made whether or not it will be enabled for data chaining
0000    406 ;                   and which of the two packet assembly routines will be used
0000    407 ;                   when a packet is to be assembled. This change results in a
0000    408 ;                   significant performance improvement whenever the controller
0000    409 ;                   is not enabled for data chaining, which is most of the time.
0000    410 ;                   This is because ASSPKT_NODCH, the routine which gets invoked
0000    411 ;                   and assumes no data chaining, does not have to do all the
0000    412 ;                   excess work that support for data chaining requires.
0000    413 ;
0000    414 ;           V03-020 TMK0001         Todd M. Katz            01-Nov-1983
0000    415 ;                   Add a performance enhancement to recieves and transmits at
0000    416 ;                   the cost of some additional space within the CDB. This
0000    417 ;                   enhancement consists of picking up the address of a ring entry
0000    418 ;                   (transmit or recieve) from the appropriate vector of such ring
0000    419 ;                   entry addresses instead of computing it. The additional space
0000    420 ;                   within the CDB consists of two vector tables - one of transmit
0000    421 ;                   ring entry addresses and the other of recieve ring entry
0000    422 ;                   addresses. These vector tables are initialized only once - when
0000    423 ;                   the CDB itself is initialized within START.
0000    424 ;
0000    425 ;           V03-019 RNG0019         Rod N. Gamache          15-Sep-1983
0000    426 ;                   Fix bug in shutdown code to allow for the first UCB
0000    427 ;                   address not to be set in the CDB (fixes bug check).
0000    428 ;                   Fix service operations that were broken by FFI interface.
0000    429 ;
0000    430 ;           V03-018 RNG0018         Rod N. Gamache          24-Aug-1983
0000    431 ;                   Fix some bugs in the Fast Interface code.
0000    432 ;                   Set the string indicator when returning NMA$C_PCLI_HWA.
0000    433 ;                   Initialize CDB Fork Block when it is allocated.
0000    434 ;
0000    435 ;           V03-017 RNG0017         Rod N. Gamache          08-Jul-1983
0000    436 ;                   Add new Fast Interface.
0000    437 ;
0000    438 ;           V03-016 RNG0016         Rod N. Gamache          06-Jul-1983
0000    439 ;                   Stop using the MBX code to create/delete UCBs.
0000    440 ;
0000    441 ;           V03-015 RNG0015         Rod N. Gamache          23-Jun-1983
0000    442 ;                   Remove internal definition of IRP$Q_STATION.
0000    443 ;
0000    444 ;           V03-014 RNG0014         Rod N. Gamache          02-May-1983
0000    445 ;                   Fix the shared protocol quota to be initialized when unit
0000    446 ;                   is inited.  Add a point-to-point startup mechanism.
0000    447 ;
0000    448 ;           V03-013 RNG0013         Rod N. Gamache          02-May-1983
0000    449 ;                   Fix the SHUT code in STARTIO to setup R4.
0000    450 ;
0000    451 ;           V03-012 RNG0012         Rod N. Gamache          30-Apr-1983
0000    452 ;                   Add code to support Point-to-Point mode.
0000    453 ;                   Cleanup shared user code and add shared user's quota.
0000    454 ;
0000    455 ;           V03-011 RNG0011         Rod N. Gamache          15-Apr-1983
0000    456 ;                   Make the NMA return codes self-describing (i.e. lonword
```

```
0000   457 ;               or strings).
0000   458 ;               Misc. cleanup.
0000   459 ;
0000   460 ;       V03-010 RNG0010         Rod N. Gamache          16-Mar-1983
0000   461 ;               Validate parameter settings for all users of a SHARED
0000   462 ;               UCB. Copy the multicast address list from the SHARED
0000   463 ;               default user's UCB to the primary UCB for the SHARED
0000   464 ;               protocol.
0000   465 ;               Process MOP requests for read counters.
0000   466 ;               Make Protocol Access a modifiable parameter.
0000   467 ;               Make CIRCUIT startup/shutdown enable/disable the protocol
0000   468 ;               type for I/O operations.
0000   469 ;
0000   470 ;       V03-009 RNG0009         Rod N. Gamache          7-Mar-1983
0000   471 ;               Wait 20 usec on successful startup, before reading the
0000   472 ;               port CSRs to give UNA time to set correct state.
0000   473 ;               Return different error codes for TIMEOUTs, and Transceiver
0000   474 ;               power cable disconnected. Also, return different error
0000   475 ;               code on XMIT operations when the device is re-starting.
0000   476 ;               Add code to accept NMA$C_PCLI_DES parameter, don't return
0000   477 ;               parameter unless user is running in LIMITED mode.
0000   478 ;               On SHUTDOWN of the promiscuous user, reset the hardware into
0000   479 ;               non-promiscuous mode.
0000   480 ;               Cleanup the SHARED-LIMITED user code.
0000   481 ;               Allocate sufficient receive buffers at FDT time to guarantee
0000   482 ;               buffers will be available at run-time.
0000   483 ;
0000   484 ;       V03-008 RNG0008         Rod N. Gamache          14-Feb-1983
0000   485 ;               Add protocol sharing to driver. Fix zeroing of driver
0000   486 ;               ccunters. Fix bug with attempting to fill receive buffer
0000   487 ;               queue when TQE timer expired.
0000   488 ;
0000   489 ;       V03-007 RNG0007         Rod N. Gamache          11-Feb-1983
0000   490 ;               Change device into promiscuous mode, when promiscuous user
0000   491 ;               starts up (even after device has already been initialized).
0000   492 ;               Change device to enable all multicasts, if the user requests
0000   493 ;               this.
0000   494 ;
0000   495 ;       V03-006 RNG0004         Rod N. Gamache          14-Jan-1983
0000   496 ;               Cleanup use of quotas. Read port status on error.
0000   497 ;               Fix counter return for driver counters. Fix even size
0000   498 ;               receive buffer problem. Set the DELMBX bit in the
0000   499 ;               UCB of units to be "deleted".
0000   500 ;
0000   501 ;--
```

```
                     ;
                     ; EXTERNAL SYMBOLS
                     ;
                             $ABDDEF                 ; Define ABDs
                             $ACBDEF                 ; Define AST control block
                             $CANDEF                 ; Define CANCEL reason codes
                             $CCBDEF                 ; Define CCB offsets
                             $CRBDEF                 ; Define CRB
                             $CXBDEF                 ; Define CXB
                             $DCDEF                  ; Define device classes and types
                             $DDBDEF                 ; Define DDB
                             $DEVDEF                 ; Define device characteristics
                             $DPTDEF                 ; Define DPT
                             $DYNDEF                 ; Define dynamic data structures
                             $FFIDEF                 ; Fast Interface definitions
                             $FKBDEF                 ; Fork block definitions
                             $IDBDEF                 ; Define IDB
                             $IODEF                  ; Define I/O function codes
                             $IPLDEF                 ; Define IPLs
                             $IRPDEF                 ; Define IRP
                             $JIBDEF                 ; Define JIB
                             $MSGDEF                 ; Define SYSTEM MESSAGES
                             $NMADEF                 ; Define Network Management Codes
                             $ORBDEF                 ; OBJECT'S RIGHTS BLOCK OFFSETS
                             $PCBDEF                 ; Define PCB
                             $PRDEF                  ; Processor register definitions
                             $PRVDEF                 ; Privilege bit definitons
                             $SSDEF                  ; Define System Status Codes
                             $TQEDEF                 ; Define TQE offsets
                             $UBADEF                 ; Define UBA symbols
                             $UCBDEF                 ; Define UCB
                             $VADEF                  ; Define Virtual Address bits
                             $VECDEF                 ; Define CRB VECTOR
                             $XMDEF                  ; Define DECnet datalink characteristics
                     ;
                     ; Local symbol definitions
                     ;

                     ;
                     ; Define the following symbol to enable use of receive buffered data paths.
                     ;       This feature cannot normally be used on systems which use UNIBUS
                     ;       disks, since the disk drivers REQUIRE the buffered data paths.
                     ;
                     ;       RCV_DATAP = 1           ; Enable use of recv buffered datapaths

                     ;
                     ; Define the following symbol to enable use of point-to-point mode
                     ;
^0000001             POINT = 1                       ; Enable use of point-to-point mode

                     ;
                     ; Argument list offsets for QIO
                     ;
00000000             P1      = 0                     ; Parameter 1
00000004             P2      = 4                     ; Parameter 2
00000008             P3      = 8                     ; Parameter 3
0000000C             P4      = 12                    ; Parameter 4
```

```
00000010  0000   560 P5      = 16                            ; Destination/Source address
          0000   561
          0000   562 ;
          0000   563 ; Constants
          0000   564 ;
          0000   565 $EQU    BRDCST1          <^XFFFFFFFF>    ; Broadcast address
          0000   566 $EQU    BRDCST2          <^XFFFF>        ;       ...
          0000   567 $EQU    UDB_SID_C_LENGTH 60              ; Size of SYSTEM ID UDBB
          0000   568 $EQU    MAX_C_MCT        10              ; Maximum number of multicast addresses
          0000   569 $EQU    MAX_C_CTR        32              ; Maximum size of counters (words)
          0000   570 $EQU    MAX_C_XMT        4               ; Maximum number of entries in XMT ring
          0000   571 $EQU    MAX_C_RCV        8               ; Maximum number of entries in RCV ring
          0000   572 $EQU    MAX_C_CHAIN      1               ; Maximum number of extra segments in
          0000   573                                         ;  a receive buffer chain
          0000   574
          0000   575 $EQU    NI_CTR_PROTYP    <^X0260>        ; Ethernet read counters protocol 60-02
          0000   576 $EQU    NI_CTR_READ      9               ; Read counters request function
          0000   577 $EQU    NI_CTR_REPLY     11              ; Read counters reply function
          0000   578
          0000   579 $EQU    INIT_C_QUOTA     9*1500          ; Allow for 9 of the largest buffers
          0000   580 $EQU    INIT_C_AQUOTA    2               ; Allow for 2 additional buffers
          0000   581 $EQU    DSC$A_POINTER    4               ; Pointer to data in buffer descriptor
          0000   582 $EQU    MIN_PKT_SIZE     46              ; Size of user data in a runt packet
          0000   583 $EQU    MAX_PKT_SIZE     1500            ; Size of maximum Ethernet user data
          0000   584 $EQU    TQE_C_DELTA      2               ; 2 second timer interval
          0000   585 $EQU    TQE_DELTA        TQE_C_DELTA*10000*1000 ; Delta interval (in 100 nsec)
          0000   586 $EQU    RESTART_DELTA    3*10000*1000    ; RESTART interval - 3 Seconds
          0000   587 $EQU    XMT_C_TIM        8               ; XMITS must take less than 8 seconds
          0000   588 $EQU    XMT_TIM          <<XMT_C_TIM+2>/2>
          0000   589 $EQU    DNI_C_TIM        12              ; DNI settings must take less than 12sec
          0000   590 $EQU    DNI_TIM          <<DNI_C_TIM+2>/2>
          0000   591 $EQU    XE_C_HEADER      14              ; Size of DEUNA/DELUA header
          0000   592 $EQU    XE_C_CNTSIZ      2               ; Size of packet count field
          0000   593 $EQU    XE_C_CRC         4               ; Size of DEUNA/DELUA CRC
          0000   594          ASSUME  XE_C_CRC LE CXB$C_TRAILER
          0000   595 $EQU    XE_C_STPRO       <^X0660>        ;% PT-TO-PT Starting protocol type
          0000   596 $EQU    IPL$_XE_FIPL     8               ; Fork IPL
          0000   597 $EQU    IPL$_XE_DIPL     21              ; Device IPL
          0000   598
          0000   599 ;
          0000   600 ; Local macros
          0000   601 ;
          0000   602 .MACRO  SETBIT  VAL,FLAG
          0000   603          .NTYPE   $$     VAL
          0000   604          .IF EQ  ?'$$'-^XOEF>
          0000   605          .IF NDF VAL
          0000   606          BBSS    S^#VAL,FLAG,.+1
          0000   607          .IFF
          0000   608          .IF LT <VAL-8>
          0000   609          BISB    #<1@VAL>,FLAG
          0000   610          .IFF
          0000   611          BBSS    #VAL,FLAG,.+1
          0000   612          .ENDC
          0000   613          .ENDC
          0000   614          .IFF
          0000   615          BBSS    VAL,FLAG,.+1
          0000   616          .ENDC
```

```
0000    617 .ENDM   SETBIT
0000    618 ;
0000    619 ; ======================
0000    620 ;
0000    621 .MACRO  CLRBIT  VAL,FLAG
0000    622         .NTYPE   $$     VAL
0000    623         .IF EQ  ?'$$'-^X0EF>
0000    624         .IF NDF VAL
0000    625         BBCC    S^#VAL,FLAG,.+1
0000    626         .IFF
0000    627         .IF LT <VAL-8>
0000    628         BICB    #<1@VAL>,FLAG
0000    629         .IFF
0000    630         BBCC    #VAL,FLAG,.+1
0000    631         .ENDC
0000    632         .ENDC
0000    633         .IFF
0000    634         BBCC    VAL,FLAG,.+1
0000    635         .ENDC
0000    636 .ENDM   CLRBIT
0000    637 ;
0000    638 ; ======================
0000    639 ;
0000    640 .MACRO  INCC    COUNTER,CONTEXT=L,?L    ; Increment counter
0000    641         INC'CONTEXT     COUNTER         ; Do Increment
0000    642         BCC     l                       ; Br if no carry set
0000    643         DEC'CONTEXT     COUNTER         ; Leave at maximum value
0000    644 l:
0000    645 .ENDM   INCC
0000    646 ;
0000    647 ; ======================
0000    648 ;
0000    649 .MACRO  CNTR    CURCNT,COUNTER,CONTEXT=L,?L  ; Accumlate counter
0000    650         ADD'CONTEXT     CURCNT,COUNTER  ; Do addition
0000    651         BCC     l                       ; Br if no carry set
0000    652         MNEGL   #1,COUNTER              ; Leave at maximum value
0000    653 l:
0000    654 .ENDM   CNTR
0000    655 ;
0000    656 ; ======================
0000    657 ;
0000    658 .MACRO  PUSHQ   ARG                     ; Push a quadword
0000    659         MOVQ    ARG,-(SP)               ; Save argument on stack
0000    660 .ENDM   PUSHQ
0000    661 ;
0000    662 ; ======================
0000    663 ;
0000    664 .MACRO  POPQ    ARG                     ; Pop a quadword
0000    665         MOVQ    (SP)+,ARG               ; Restore argument
C000    666 .ENDM   POPQ
0000    667 ;
0000    668 ; ======================
0000    669 ;
0000    670 .MACRO  PARAM   TYPE,OFFSET,WIDTH,MIN,MAX,INVALID,BASE=UCB,STRING,-
0000    671                 SIZE,CHECK=YES
0000    672 ;
0000    673 ;       Macro to generate the parameter tables
```

```
0000    674 ;
0000    675 ; Inputs:
0000    676 ;
0000    677 ;       TYPE = Parameter type
0000    678 ;       OFFSET = Offset in UCB/CDB to current value
0000    679 ;       WIDTH = Width of field in UCB/CDB (B,W,L)
0000    680 ;       MIN = Minimum value parameter is allowed to take
0000    681 ;       MAX = Maximum value parameter is allowed to take
0000    682 ;       INVALID = Invalid flags in status word
0000    683 ;       BASE = Data base (CDB,UCB)
0000    684 ;       STRING = Parameter is a string value
0000    685 ;       SIZE = Maximum size of string parameter in bytes
0000    686 ;       CHECK = Comparison is needed (YES,NO)
0000    687 ;
0000    688         .IF BLANK type
0000    689          .WORD  0
0000    690         .IF_FALSE       ; BLANK type
0000    691          $$$typ = type & prm_typ_m_code         ; Isolate type code
0000    692
0000    693          $$$flg = 0
0000    694          .IIF NOT_BLANK <invalid>, $$$flg = $$$flg!prm_flg_m_invalid
0000    695          .IIF IDN <check><YES>, $$$flg = $$$flg!prm_flg_m_check
0000    696          .IIF IDN <base><CDB>, $$$flg = $$$flg!prm_flg_m_cdb
0000    697
0000    698          .IF BLANK string
0000    699
0000    700                .WORD   $$$typ
0000    701           .IIF NOT_BLANK        <min>, $$$flg = $$$flg!prm_flg_m_min
0000    702           .IIF NOT_BLANK        <max>, $$$flg = $$$flg!prm_flg_m_max
0000    703                .BYTE   $$$flg
0000    704           $$$off = offset & prm_off_m_value     ; Isolate offset only
0000    705           $$$wid = 0                            ; Set null width
0000    706           .IIF IDN <width><B>, $$$wid = <1@prm_off_v_width>
0000    707           .IIF IDN <width><W>, $$$wid = <2@prm_off_v_width>
0000    708           .IIF IDN <width><L>, $$$wid = <3@prm_off_v_width>
0000    709                .WORD   $$$off!$$$wid
0000    710           .IIF NOT_BLANK        <min>, .WORD    min
0000    711           .IIF NOT_BLANK        <max>, .WORD    max
0000    712           line_prm_bufsiz = line_prm_bufsiz + 6
0000    713
0000    714          .IF_FALSE      ; BLANK STRING
0000    715
0000    716                .WORD   $$$typ!prm_typ_m_string ; Indicate a string parameter
0000    717                .BYTE   $$$flg
0000    718            $$$off = offset & prm_off_m_value    ; Isolate offset only
0000    719            $$$wid = <size @ prm_off_v_width> & prm_off_m_width ; Get max allowed
0000    720            $$$siz = <$$$wid @ -prm_off_v_width>
0000    721                .WORD   $$$off!$$$wid
0000    722            line_prm_bufsiz = line_prm_bufsiz + 4 + $$$siz
0000    723
0000    724          .ENDC          ; BLANK STRING
0000    725
0000    726          .IIF NOT_BLANK <invalid>, .WORD    invalid
0000    727
0000    728         .ENDC           ; BLANK TYPE
0000    729 .ENDM   PARAM
0000    730 ;
```

```
0000    731 ; ======================
0000    732 ;
0000    733 .MACRO  COUNTER TYPE,WIDTH=16,OFFSET=0,BITMAP,BASE=LINE
0000    734         .IIF NDF 'base'_cnt_size, 'base'_cnt_size = 0
0000    735         .IIF NDF 'base'_cnt_bufsiz, 'base'_cnt_bufsiz = 0
0000    736         $$$typ = nma$c_ctlin_'type' & nma$m_cnt_typ
0000    737         .IIF IDN <bitmap><MAP>, $$$typ = $$$typ!<nma$m_cnt_map>
0000    738         $$$wid = 0                          ; Set reserved mask width
0000    739         .IIF IDN <width><8>, $$$wid = <1@nma$v_cnt_wid>
0000    740         .IIF IDN <width><16>, $$$wid = <2@nma$v_cnt_wid>
0000    741         .IIF IDN <width><32>, $$$wid = <3@nma$v_cnt_wid>
0000    742         .IIF EQ $$$wid, .ERROR              ; Invalid bit width value
0000    743         .WORD   nma$m_cnt_cou!$$$wid!$$$typ
0000    744         $$$off = offset/<width/8>
0000    745         .WORD   $$$off
0000    746         'base'_cnt_size = 'base'_cnt_size + 1   ; Tally one more entry
0000    747         'base'_cnt_bufsiz = 'base'_cnt_bufsiz + 2 + <width/8>
0000    748         .IIF IDN <bitmap><MAP>, 'base'_cnt_bufsiz = 'base'_cnt_bufsiz + 2
0000    749 .ENDM   COUNTER
0000    750 ;
0000    751 ; ======================
0000    752 ;
0000    753 .MACRO  OFSET   SIZE,OFFSET,BASE
0000    754         .IF IDN <base><LINE>
0000    755         .WORD   cdb_'size'_'offset'
0000    756         .IFF
0000    757         .WORD   ucb$'size'_xe_'offset'
0000    758         .ENDC
0000    759 .ENDM   OFSET
0000    760 ;
0000    761 ; ======================
0000    762 ;
0000    763 .MACRO  DRVRCTR TYPE,WIDTH=16,OFFSET=0,BASE=LINE
0000    764         .IIF NDF 'base'_ctr_size, 'base'_ctr_size = 0
0000    765         .IIF NDF 'base'_ctr_bufsiz, 'base'_ctr_bufsiz = 0
0000    766         .IF IDN <base><LINE>
0000    767         $$$typ = nma$c_ctlin_'type' & nma$m_cnt_typ
0000    768         .IFF
0000    769         $$$typ = nma$c_ctcir_'type' & nma$m_cnt_typ
0000    770         .ENDC
0000    771         $$$wid = 0                          ; Set reserved mask width
0000    772         .IIF IDN <width><8>, $$$wid = <1@nma$v_cnt_wid>
0000    773         .IIF IDN <width><16>, $$$wid = <2@nma$v_cnt_wid>
0000    774         .IIF IDN <width><32>, $$$wid = <3@nma$v_cnt_wid>
0000    775         .IIF EQ $$$wid, .ERROR              ; Invalid bit width value
0000    776         .WORD   nma$m_cnt_cou!$$$wid!$$$typ
0000    777         .IIF IDN <width><8>, OFSET B,'offset','base
0000    778         .IIF IDN <width><16>, OFSET W,'offset','base
0000    779         .IIF IDN <width><32>, OFSET L,'offset','base
0000    780         'base'_ctr_size = 'base'_ctr_size + 1   ; Tally one more entry
0000    781         'base'_ctr_bufsiz = 'base'_ctr_bufsiz + 2 + <width/8>
0000    782 .ENDM   DRVRCTR
0000    783 ;
0000    784 ; ======================
0000    785 ;
0000    786 .MACRO  MOPCTR  WIDTH=16,OFFSET,BITMAP
0000    787         .IIF NDF mop_ctr_size, mop_ctr_size = 0
```

```
0000    788          $$$map = 0
0000    789          $$$wid = width/8
0000    790          .IIF IDN <bitmap><MAP>, $$$map = 1@7
0000    791          .IF NOT_BLANK   <offset>
0000    792                  .BYTE   offset          ; Offset in couter table
0000    793                  .BYTE   $$$map!$$$wid   ; Counter width in bytes + BITMAP FLAG
0000    794                  mop_ctr_size = mop_ctr_size + $$$wid
0000    795          .IFF ; NOT_BLANK
0000    796                  .WORD   0               ; End of table
0000    797          .ENDC ; NOT_BLANK
0000    798  .ENDM   MOPCTR
0000    799  ;
0000    800  ; ======================
0000    801  ;
0000    802  .MACRO  SKIP    BIT,LOC,REG,CONTEXT=W,?L        ; SKIP FIELD
0000    803          BBC     #BIT,LOC,L              ; Br if field not present
0000    804          TST'CONTEXT     (REG)+          ; Skip next field
0000    805  L:
0000    806  .ENDM   SKIP
0000    807  ;
0000    808  ; ======================
0000    809  ;
0000    810  .MACRO  $DISPATCH,      INDX,VECTOR,TYPE=W,NMODE=S^#,?MN,?MX,?S,?SS,?ZZ
0000    811  SS:
0000    812
0000    813          .MACRO  $DSP1,$DSP1_1
0000    814                  .IRP    $DSP1_2,$DSP1_1
0000    815                  $DSP2   $DSP1_2
0000    816                  .ENDR
0000    817          .ENDM   $DSP1
0000    818
0000    819          .MACRO  $DSP2,$DSP2_1,$DSP2_2
0000    820                  .=<$DSP2_1-MN>*2 + S
0000    821                  .WORD   $DSP2_2-S
0000    822          .ENDM   $DSP2
0000    823
0000    824
0000    825          .MACRO  $BND1,$BND1_1,$BND1_2,$BND1_3
0000    826                  $BND2   $BND1_1,$BND1_2
0000    827          .ENDM   $BND1
0000    828
0000    829          .MACRO  $BND2,$BND2_1,$BND2_2
0000    830                  .IIF    $BND2_1,$BND2_2-.,      .=$BND2_2
0000    831          .ENDM   $BND2
0000    832
0000    833          .MACRO  $BND    $BND_1,$BND_2
0000    834                  .IRP    $BND_3,<$BND_2>
0000    835                  $BNDT   $BND_1,$BND_3
0000    836                  .ENDR
0000    837          .ENDM   $BND
0000    838
0000    839          .=0
0000    840  ZZ:
0000    841          $BND    GT,<VECTOR>
0000    842  MX:
0000    843          $BND    LT,<VECTOR>
0000    844  MN:
```

```
0000    845          .=SS
0000    846
0000    847 CASE'TYPE        INDX,#<MN-ZZ>,NMODE'<MX-MN>
0000    848 S:
0000    849          .REPT    MX-MN+1
0000    850          .WORD    <MX-MN>*2 + 2
0000    851          .ENDR
0000    852
0000    853          .=S
0000    854
0000    855          $DSP1    <<VECTOR>>
0000    856
0000    857          .=<MX-MN>*2 + S + 2
0000    858
0000    859 .ENDM    $DISPATCH
0000    860
```

```
                     0000   862 ;
                     0000   863 ; Overlays of IRP
                     0000   864 ;
                     0000   865         $DEFINI IRP     GLOBAL
                     0000   866
00000021             0000   867 . = IRP$W_FUNC+1
                     0021   868 $DEF     IRP$B_XE_FUNC   .BLKB   1       ; DEUNA/DELUA driver internal function code
                     0022   869
00000038             0022   870 . = IRP$L_MEDIA
                     0038   871 $DEF     IRP$W_XE_RID                    ; RCV/XMT request ID
                     0038   872 $DEF     IRP$B_XE_SLOT   .BLKB   1       ; RCV/XMT mapping slot number
                     0039   873 $DEF     IRP$B_XE_RING   .BLKB   1       ; RCV/XMT ring entry number
                     003A   874
00000046             003A   875 . = IRP$Q_STATION+6
                     0046   876 $DEF     IRP$B_XE_DATAP  .BLKB   1       ; XMT buffered data path number
                     0047   877
0000003C             0047   878 . = IRP$L_MEDIA+4
                     003C   879 $DEF     IRP$L_XE_SYSBUF .BLKL   1       ; XMT system buffer address
                     0040   880
0000003C             0040   881 . = IRP$L_MEDIA+4
                     003C   882 $DEF     IRP$L_XE_DATBUF .BLKL   1       ; User RCV data buffer address
                     0040   883
00000038             0040   884 . = IRP$L_MEDIA
                     0038   885 $DEF     IRP$W_XE_USERSIZ .BLKW  1       ; User P2 buffer size on sensemode
                     003A   886
0000003A             003A   887 . = IRP$L_MEDIA+2
                     003A   888 $DEF     IRP$W_XE_STATUS .BLKW   1       ; Completion status
                     003C   889
0000003C             003C   890 . = IRP$L_MEDIA+4
                     003C   891 $DEF     IRP$L_XE_USERBUF .BLKL  1       ; User P1 buffer address on sensemode
                     0040   892
00000040             0040   893 . = IRP$Q_STATION
                     0040   894 $DEF     IRP$L_XE_P2BUF  .BLKL   1       ; User P2 buffer address on sensemode
                     0044   895
00000044             0044   896 . = IRP$Q_STATION+4
                     0044   897 $DEF     IRP$W_XE_P2SIZ  .BLKW   1       ; P2 return buffer size on sensemode
                     0046   898
00000040             0046   899 . = IRP$Q_STATION
                     0040   900 $DEF     IRP$W_XE_CODE   .BLKW   1       ; Bad parameter code on startup request
                     0042   901
0000003C             0042   902 . = IRP$L_MEDIA+4
                     003C   903 $DEF     IRP$L_XE_MAP    .BLKL   1       ; Diagnostics buffer mapping info
                     0040   904
00000040             0040   905 . = IRP$Q_STATION
                     0040   906 $DEF     IRP$L_XE_DGUNI  .BLKL   1       ; Diagnostics buffer UNIBUS address
                     0044   907 $DEF     IRP$L_XE_UPADR  .BLKW   1       ; Micro-process internal address
                     0046   908
00000094             0046   909 . = IRP$L_RBUFH_AD
                     0094   910         ASSUME  IRP$L_RBOFF EQ .+4
                     0094   911 $DEF     IRP$Q_XE_PCB    .BLKQ   1       ; Snap shot of PCBB
                     009C   912
0000003A             009C   913 . = IRP$L_MEDIA+2
                     003A   914 $DEF     IRP$W_XE_PROTYP .BLKW   1       ; Protocol type for user
                     003C   915
00000090             003C   916 . = IRP$L_LBOFF
                     0090   917 $DEF     IRP$L_XE_SHR    .BLKL   1       ; Address of SHR structure for user
                     0094   918
```

```
00000060  0094   919 . = IRP$L_FQFL
          0060   920 $DEF     IRP$C_XE_STD    .BLKL   1       ; End of "standard" IRP
          0064   921
          0064   922 ;
          0064   923 ; Define driver internal function codes stored in IRP$B_XE_FUNC of IRP.
          0064   924 ; NOTE: These are not really used as bit offsets - but as values.
          0064   925 ;
          0064   926         _VIELD  XE_FC,0,<-              ; Internal function codes
          0064   927                 <INIT>,-                ; Initialize DEUNA/DELUA
          0064   928                 <TESTD>,-               ; Perform startup self test
          0064   929                 <LDRING>,-              ; Load ring descriptors
          0064   930                 <WTPHYAD>,-             ; Write physical address
          0064   931                 <WTMULTI>,-             ; Write multicast address list
          0064   932                 <WTMODE>,-              ; Write mode
          0064   933                 <START>,-               ; Start XMIT/RECV processes
          0064   934                 <XMIT>,-                ; Transmit request
          0064   935                 <RECV>,-                ; Receive request
          0064   936                 <STOP>,-                ; Stop protocol
          0064   937                 <RDCNTS>,-              ; Read counters
          0064   938                 <RCCNTS>,-              ; Read and clear counters
          0064   939                 <MULTI>,-               ; Write new multicast address list
          0064   940                 <LDMEM>,-               ; Load memory
          0064   941                 <RDMEM>,-               ; Read memory
          0064   942                 <UPST>,-                ; Load and start micro-address
          0064   943                 <LDSID>,-               ; Load system ID parameters
          0064   944                 <RDSID>,-               ; Read system ID parameters
          0064   945                 <STOPPR>,-              ; Stop xmit and receive process
          0064   946                 <STEST>,-               ; Perform a self test
          0064   947                 <READST>,-              ; Read port status
          0064   948                 <RDDPA>,-               ; Read default physical address
          0064   949                 <CANCEL>,-              ; Cancel request
          0064   950                 <RESTART>,-             ; Restart PROTOCOL
          0064   951                 <CHMODE>,-              ; Change the mode definition
          0064   952                 <MOPCTR>,-              ; MOP read counters request
          0064   953                 <ESTEST>,-              ; Extended Selt-test (DELUA Only)
          0064   954                 <HALT>,-                ; Halt Port (DELUA Only)
          0064   955                 >                       ;
          0064   956
          0064   957         $DEFEND IRP                     ; End of IRP overlays
```

```
                     0000   959 ;
                     0000   960 ; Overlays of CXB
                     0000   961 ;
                     0000   962         $DEFINI CXB     GLOBAL
                     0000   963
00000020             0000   964 . = CXB$L_SPARE1
                     0020   965 $DEF     CXB$B_XE_FUNC   .BLKB   1       ; DEUNA/DELUA driver internal function code
00000022             0021   966                         .BLKB   1       ; SPARE
                     0022   967 $DEF     CXB$W_XE_RID                    ; RCV/XMT request ID
                     0022   968 $DEF     CXB$B_XE_SLOT   .BLKB   1       ; RCV/XMT mapping slot number
                     0023   969 $DEF     CXB$B_XE_RING   .BLKB   1       ; RCV/XMT ring entry number
                     0024   970
                     0024   971 ;
                     0024   972 ; The following overlays are for transmits only
                     0024   973 ;
00000024             0024   974 . = CXB$L_SPARE0
                     0024   975 $DEF     CXB$L_T_IRP     .BLKL   1       ; Associated IRP address (if any)
                     0028   976
0000003A             0028   977 . = CXB$C_HEADER-<XE_C_HEADER>
                     003A   978 $DEF     CXB$T_T_DATA    .BLKB   XE_C_HEADER ; Standard Ethernet header
                     0048   979
                     0048   980 ;
                     0048   981 ; The following overlays are for receives only
                     0048   982 ;
0000000B             0048   983 . = CXB$B_CODE
                     000B   984 $DEF     CXB$B_R_FLAGS   .BLKB   1       ; Receive message flags
                     000C   985
0000001C             000C   986 . = CXB$L_END_ACTION
                     001C   987 $DEF     CXB$W_R_NCHAIN  .BLKW   1       ; Number of buffers in cha n
                     001E   988
00000014             001E   989 . = CXB$L_IRP
                     0014   990 $DEF     CXB$W_R_LENERR  .BLKW   1       ; Length and receive error status
                     0016   991
00000038             0016   992 . = CXB$C_HEADER - <XE_C_HEADER+XE_C_CNTSIZ>
                     0038   993 $DEF     CXB$T_R_DATA                    ; Start of receive data
                     0038   994 $DEF     CXB$G_R_DEST    .BLKW   3       ; Destination node address
                     003E   995 $DEF     CXB$G_R_SRC     .BLKW   3       ; Source node address
                     0044   996 $DEF     CXB$W_R_PTYPE   .BLKW   1       ; Protocol Type
                     0046   997 $DEF     CXB$T_R_USERDAT                 ; Start of user data
                     0046   998 $DEF     CXB$W_R_SIZE    .BLKW   1       ; Size of received message (if padded).
                     0048   999
                     0048  1000 ;
                     0048  1001 ; NOTE: The CXB functions are the same as for an IRP (IRP$B_XE_FUNC)
                     0048  1002 ;
                     0048  1003
                     0048  1004         $DEFEND CXB                     ; End of CXB overlays
```

```
                     0000  1006 ;
                     0000  1007 ; Definitions that follow the standard UCB fields
                     0000  1008 ;
                     0000  1009
                     0000  1010          $DEFINI UCB     GLOBAL          ; Start of UCB definitions
                     0000  1011
00000098             0000  1012 . = UCB$C_NI_LENGTH                      ; Position at end of UCB NI extension
                     0098  1013
                     0098  1014 $DEF     UCB$Q_XE_QUEUES                 ; Message and I/O request queue heads
                     0098  1015 $DEF     UCB$Q_XE_SHARE  .BLKQ   1       ; List of shared users
                     00A0  1016 $DEF     UCB$Q_XE_IOQS                   ; Start of the I/O queues
                     00A0  1017 $DEF     UCB$Q_XE_RCVMSG .BLKQ   1       ; Receive messages completed
                     00A8  1018 $DEF     UCB$Q_XE_RCVREQ .BLKQ   1       ; Receive IRP waiting for messages
                     00B0  1019 $DEF     UCB$Q_XE_XMTREQ .BLKQ   1       ;% Xmit IRP wait queue (PT-TO-PT)
00000004             00B8  1020 UCB$C_XE_QUEUES = <.-UCB$Q_XE_QUEUES>/8 ; Number of queue heads
                     00B8  1021
                     00B8  1022 $DEF     UCB$L_XE_PID    .BLKL   1       ; Starter's PID
                     00BC  1023 $DEF     UCB$L_XE_CPID   .BLKL   1       ; Creator's PID
                     00C0  1024 $DEF     UCB$L_XE_AST    .BLKL   1       ; Attention AST list
                     00C4  1025 $DEF     UCB$L_XE_DEFUSR .BLKL   1       ; Default shared user (shared use only)
                     00C8  1026 $DEF     UCB$W_XE_QUOTA  .BLKW   1       ; Receive buffer quota
                     00CA  1027 $DEF     UCB$W_XE_PROTYP .BLKW   1       ; Ethernet protocol type
                     00CC  1028
                     00CC  1029 $DEF     UCB$B_XE_SETPRM                 ; Start of parameter section
                     00CC  1030 $DEF     UCB$G_XE_DES    .BLKW   3       ; Destination address for shared user
                     00D2  1031 $DEF     UCB$W_XE_HBQ    .BLKW   1       ; Hardware buffer quota
                     00D4  1032 $DEF     UCB$B_XE_ACC    .BLKB   1       ; Protocol access mode
                     00D5  1033 $DEF     UCB$B_XE_BFN    .BLKB   1       ; Number of receive buffers
                     00D6  1034
                     00D6  1035 $DEF     UCB$B_XE_SHRPRM                 ; Start of "shared user" validated prms
                     00D6  1036 $DEF     UCB$W_XE_BSZ    .BLKW   1       ; Device buffer size
                     00D8  1037 $DEF     UCB$B_XE_PRO    .BLKB   1       ; Protocol selection
                     00D9  1038 $DEF     UCB$B_XE_PAD    .BLKB   1       ; Padding mode
                     00DA  1039 $DEF     UCB$B_XE_DCH    .BLKB   1       ; Data chaining
                     00DB  1040 $DEF     UCB$B_XE_PRM    .BLKB   1       ; Promiscuous mode
                     00DC  1041 $DEF     UCB$B_XE_MLT    .BLKB   1       ; Multicast (all) address state
                     00DD  1042
                     00DD  1043 $DEF     UCB$B_XE_CDBPRM                 ; Start of settable parameters for CDB
                     00DD  1044 $DEF     UCB$B_XE_CRC    .BLKB   1       ; CRC generation mode
                     00DE  1045 $DEF     UCB$B_XE_CON    .BLKB   1       ; Controller mode
                     00DF  1046 $DEF     UCB$B_XE_EKO    .BLKB   1       ; Echo mode (DEUNA Only)
                     00E0  1047 $DEF     UCB$B_XE_ILP    .BLKB   1       ; Internal Loopback mode (DELUA Only)
00000004             00E1  1048 UCB$C_XE_CDBPRM = .-UCB$B_XE_CDBPRM
0000000B             00E1  1049 UCB$C_XE_SHRPRM = .-UCB$B_XE_SHRPRM
                     00E1  1050
                     00E1  1051 $DEF     UCB$G_XE_PHA    .BLKW   3       ; User defined physical address
0000000A             00E7  1052 UCB$C_XE_SETPRM = .-UCB$B_XE_CDBPRM
                     00E7  1053
                     00E7  1054 $DEF     UCB$B_XE_MULTI  .BLKB   1       ; Number of entries in MULTI
                     00E8  1055 $DEF     UCB$B_XE_MLTTBL .BLKB   1       ; Number of entries in MLTTBL
                     00E9  1056 $DEF     UCB$G_XE_MULTI  .BLKW   3*MAX_C_MLT ; Multicast address list
                     0125  1057 $DEF     UCB$G_XE_MLTTBL .BLKW   3*MAX_C_MLT ; Multicast generation list
                     0161  1058
                     0161  1059 $DEF     UCB$W_XE_CTR                    ; Start of counter section
                     0161  1060 $DEF     UCB$W_XE_MNECTR .BLKW   1       ; Multicast address not enabled
                     0163  1061 $DEF     UCB$W_XE_UBUCTR .BLKW   1       ; No buffer available counter
                     0165  1062 $DEF     UCB$L_XE_SBLCTR .BLKL   1       ; Number of blocks sent
```

```
0169  1063 $DEF    UCB$L_XE_SBYCTR .BLKL   1       ; Number of bytes sent
016D  1064 $DEF    UCB$L_XE_RBLCTR .BLKL   1       ; Number of blocks received
0171  1065 $DEF    UCB$L_XE_RBYCTR .BLKL   1       ; Number of bytes received
0175  1066                                         ; Unused/unneeded fields
0175  1067 $DEF    UCB$W_XE_TOTQUO .BLKW   1       ; Total quota for shared UCB
0177  1068 $DEF    UCB$B_XE_MST    .BLKB   1       ; Maintenance state
0178  1069 $DEF    UCB$L_XE_FFI    .BLKL   1       ; Fast interface BLOCK address
017C  1070 $DEF    UCB$L_XE_STIRP  .BLKL   1       ;% Address of PT-TO-PT Startup IRP
0180  1071 $DEF    UCB$L_XE_CDB    .BLKL   1       ; Address of CDB
0184  1072
0184  1073 $DEF    UCB$C_XE_LENGTH                 ; Size of XEDRIVER UCB
0184  1074
0184  1075 ;
0184  1076 ; Define device status bits
0184  1077 ;
0184  1078         $VIELD  UCB,0,<-                ; XEDRIVER UCB$W_DEVSTS bits
0184  1079                 <XE_INITED,,M>,-        ; Device is initialized
0184  1080                 <,15>,-                 ; RESERVED
0184  1081                 <XE_PROTYP,,M>,-        ; Protocol type specified
0184  1082                 <XE_SHARE,,M>,-         ; Shared protocol type
0184  1083                 <XE_RUN,,M>,-           ; Unit is in RUN mode
0184  1084                 <XE_START,,M>,-         ;% Unit is in PT-TO-PT startup state
0184  1085                 <XE_STACK,,M>,-         ;% Unit is in PT-TO-PT stack state
0184  1086                 <,7>,-                  ; Reserved
0184  1087                 <XE_INTERLOCK,,M>,-     ; RESTART bit is interlocked
0184  1088                 <XE_RESTART,,M>,-       ; Automatic RESTART on PROTOCOL
0184  1089                 >                       ;  requested
0184  1090
0184  1091         $DEFEND UCB                     ; End of UCB definitions
```

```
0000  1093 ;
0000  1094 ; Device register offsets and bit definitions
0000  1095 ;
0000  1096          $DEFINI XE      GLOBAL          ; Start of port CSR definitions
0000  1097
0000  1098 $DEF     PCSR0   .BLKW   1               ; 1st word of port CSRs
0002  1099
0002  1100          _VIELD  XE_PCSR0,0,<-           ; PCSR0 bit definitions
0002  1101                  <COMAND,4,M>,-          ; Port command
0002  1102                  <ILLINT,,M>,-           ; Illegal Interrupt (SOFTWARE DEFINED)
0002  1103                  <RSET,,M>,-             ; DEUNA/DELUA Reset
0002  1104                  <INTE,,M>,-             ; Interrupt enable
0002  1105                  <INTR,,M>,-             ; Interrupt summary
0002  1106                  <USCI,,M>,-             ; Unsolicitated State Change interrupt
0002  1107                  <FATL,,M>,-             ; Fatal Error interrupt (DELUA Only)
0002  1108                  <RCBI,,M>,-             ; Receive buffer unavailable interrupt
0002  1109                  <DNI,,M>,-              ; Done interrupt
0002  1110                  <TXI,,M>,-              ; Transmit done interrupt
0002  1111                  <RXI,,M>,-              ; Receive done interrupt
0002  1112                  <PCEI,,M>,-             ; Port command error interupt
0002  1113                  <SERI,,M>,-             ; Status Error interrupt
0002  1114                  >
0002  1115
0002  1116          _VIELD  XE_CMD,0,<-             ; Define port commands
0002  1117                  <NOOP>,-                ; NO-OP
0002  1118                  <GETPCBB>,-             ; Get Port Control Block Base (PCBB)
0002  1119                  <GETCMD>,-              ; Get command from PCBB
0002  1120                  <STEST>,-               ; Perform self test
0002  1121                  <START>,-               ; Reset xmit and receive ring pointers
0002  1122                  <BOOT>,-                ; Boot DEUNA/DELUA
0002  1123                  <,2>,-                  ; RESERVED
0002  1124                  <PDMD>,-                ; Polling demand wake up
0002  1125                  <,2>,-                  ; RESERVED
0002  1126                  <ESTEST>,-              ; Extended Self-test (DELUA Only)
0002  1127                  <,2>,-                  ; RESERVED
0002  1128                  <HALT>,-                ; Halt Port (DELUA Only)
0002  1129                  <STOP>,-                ; Suspend the DEUNA/DELUA
0002  1130                  >
0002  1131
0002  1132 $DEF     PCSR1   .BLKW   1               ; 2nd word of port CSRs
0004  1133
0004  1134          _VIELD  XE_PCSR1,0,<-           ; PCSR1 bit definitions
0004  1135                  <STATE,4,M>,-           ; State
0004  1136                  <DEVID,3,M>,-           ; Device Identification
0004  1137                  <PCTO,1,M>,-            ; Port command timed out
0004  1138                  <SELF_TEST,6,M>,-       ; Selftest error code
0004  1139                  <ICAB,,M>,-             ; Interconnect Cable BAD (DEUNA Only)
0004  1140                  <XPWR,,M>,-             ; Transceiver Power BAD (DEUNA Only)
0004  1141                  >
0004  1142
0004  1143          _VIELD  XE_STAT,0,<-            ; Define port states
0004  1144                  <RESET>,-               ; Reset
0004  1145                  <PRIMLD>,-              ; Primary load
0004  1146                  <READY>,-               ; Ready
0004  1147                  <RUN>,-                 ; Running
0004  1148                  <,1>,-                  ; RESERVED
0004  1149                  <UHALT>,-               ; UNIBUS halted
```

```
0004  1150                    <NIHALT>,-              ; NI halted
0004  1151                    <NIUHALT>,-             ; NI and UNIBUS halted
0004  1152                    <PHALT>,-               ; Port halted (DELUA Only)
0004  1153                    >
0004  1154
0004  1155            _VIELD  XE_DEVID,0,<-           ; Define device identification codes
0004  1156                    <DEUNA>,-               ; DEUNA
0004  1157                    <DELUA>,-               ; DELUA
0004  1158                    >
0004  1159
0004  1160 $DEF     PCSR2                           ; 3rd word of port CSRs
0004  1161 $DEF     PCBB0   .BLKW   1               ; Low 16 bits of the Port Control Block
0006  1162                                          ;& base
0006  1163
0006  1164 $DEF     PCSR3                           ; 4th word of port CSRs
0006  1165 $DEF     PCBB2   .BLKW   1               ; Hi 2 bits of the Port Control Block
0008  1166                                          ;& base
0008  1167
0008  1168          $DEFEND XE                      ; End of device register definitions
```

```
                     0000  1170
                     0000  1171 ;
                     0000  1172 ; Define the Port Control Block Base (PCBB) Offsets and Function codes
                     0000  1173 ;
                     0000  1174          $DEFINI PBB     GLOBAL          ; Define Port Control Block Base
                     0000  1175
                     0000  1176 $DEF     PBB_B_FUNC      .BLKB   1       ; PCBB function code
                     0001  1177 $DEF     PBB_B_SUBFUNC   .BLKB   1       ; PCBB sub function code
                     0002  1178 $DEF     PBB_W_PCBB2     .BLKW   1       ; Command specific
                     0004  1179 $DEF     PBB_W_PCBB4     .BLKW   1       ;&
                     0006  1180 $DEF     PBB_W_PCBB6     .BLKW   1       ;&
                     0008  1181 $DEF     PBB_C_LENGTH                    ; Length of PBB
          00000008   0008  1182 PBB_K_LENGTH = PBB_C_LENGTH              ; Length of PBB
                     0008  1183
                     0008  1184          _VIELD  PB_FC,0,<-              ; Port Control Block function codes
                     0008  1185                  <NOOP>,-                ; NO-OP
                     0008  1186                  <LSUADDR>,-             ; Load and start microaddress
                     0008  1187                  <RDDEFAULT>,-           ; Read default physical address
                     0008  1188                  <,1>,-                  ; RESERVED
                     0008  1189                  <RDPHYAD>,-             ; Read physical address
                     0008  1190                  <WTPHYAD>,-             ; Write physical address
                     0008  1191                  <RDMULTI>,-             ; Read multicast address list
                     0008  1192                  <WTMULTI>,-             ; Write multicast address list
                     0008  1193                  <RDRING>,-              ; Read ring format
                     0008  1194                  <WTRING>,-              ; Write ring format
                     0008  1195                  <RDCNTS>,-              ; Read counters
                     0008  1196                  <RCCNTS>,-              ; Read and clear counters
                     0008  1197                  <RDMODE>,-              ; Read mode
                     0008  1198                  <WTMODE>,-              ; Write mode
                     0008  1199                  <RDSTATUS>,-            ; Read port status
                     0008  1200                  <RCSTATUS>,-            ; Read and clear port status
                     0008  1201                  <DUMPMEM>,-             ; Dump internal memory
                     0008  1202                  <LOADMEM>,-             ; Load internal memory
                     0008  1203                  <RDSYSID>,-             ; Read system ID parameters
                     0008  1204                  <WTSYSID>,-             ; write system ID paramteters
                     0008  1205                  <RDSERAD>,-             ; Read load server address
                     0008  1206                  <WTSERAD>,-             ; Write load server address
                     0008  1207                  >
                     0008  1208
                     0008  1209          _VIELD  PB_MOD,0,<-             ; Port control block mode definitions
                     0008  1210                  <HDX,,M>,-              ; Half duplex mode (Echo mode - DEUNA)
                     0008  1211                  <,1>,-                  ; RESERVED
                     0008  1212                  <LOOP,,M>,-             ; Enable internal Loopback
                     0008  1213                  <DTCR,,M>,-             ; Disables CRC generation on transmit
                     0008  1214                  <,2>,-                  ; RESERVED
                     0008  1215                  <INTL,,M>,-             ; Enable Internal Loopback (DELUA Only)
                     0008  1216                  <,2>,-                  ; RESERVED
                     0008  1217                  <DMNT,,M>,-             ; Disable maintenance features
                     0008  1218                  <,1>,-                  ; RESERVED
                     0008  1219                  <ECT,,M>,-              ; Enable H4000 (xcvr) collision test
                     0008  1220                  <TPAD,,M>,-             ; Enables transmission of PAD characters
                     0008  1221                  <DRDC,,M>,-             ; Disables DATA CHAINING on receives
                     0008  1222                  <ENAL,,M>,-             ; Enables recognition of all MULTICASTS
                     0008  1223                  <PROM,,M>,-             ; Enables promiscuous mode
                     0008  1224                  >
                     0008  1225
                     0008  1226          $DEFEND PBB                     ; End of PCBB definitions
```

```
                     0000  1228 ;
                     0000  1229 ; Define Unibus Data Block Base (UDBB) for ring buffers
                     0000  1230 ;
                     0000  1231         $DEFINI UDB     GLOBAL
                     0000  1232
                     0000  1233 $DEF     UDB_BUF_W_TDRBL .BLKW   1       ; Transmit Desc Ring Base low 16 bits
                     0002  1234 $DEF     UDB_BUF_B_TDRBH .BLKB   1       ; Transmit Desc Ring Base high 2 bits
                     0003  1235 $DEF     UDB_BUF_B_TELEN .BLKB   1       ; Length of each transmit entry
                     0004  1236 $DEF     UDB_BUF_W_TRLEN .BLKW   1       ; Number of entries in the XMIT desc ring
                     0006  1237 $DEF     UDB_BUF_W_RDRBL .BLKW   1       ; Receive Desc Ring Base low 16 bits
                     0008  1238 $DEF     UDB_BUF_B_RDRBH .BLKB   1       ; Receive Desc Ring Base hi 2 bits
                     0009  1239 $DEF     UDB_BUF_B_RELEN .BLKB   1       ; Length of each receive entry
                     000A  1240 $DEF     UDB_BUF_W_RRLEN .BLKW   1       ; Number of entries in the RECV desc ring
                     000C  1241 $DEF     UDB_BUF_C_LENGTH                ; Size of UDBB for ring buffers
0000000C             000C  1242 UDB_BUF_K_LENGTH = UDB_BUF_C_LENGTH      ; Size of UDBB for ring buffers
                     000C  1243
                     000C  1244 ;
                     000C  1245 ; Define the Unibus Data Block Base (UDBB) for getting counters
                     000C  1246 ;
00000000             000C  1247 .=0
                     0000  1248 $DEF     UDB_CTR_L_ENTRY .BLKW   1       ; Each entry is one word
                     0002  1249 $DEF     UDB_CTR_C_LENGTH                ; Size of each counter entry
00000040             0002  1250 UDB_CTR_K_LENGTH = UDB_CTR_C_LENGTH * MAX_C_CTR ; Size of counter area
                     0002  1251
                     0002  1252 ;
                     0002  1253 ; Define the Unibus Data Block Base (UDBB) for a Multicast address list
                     0002  1254 ;
00000000             0002  1255 .=0
                     0000  1256 $DEF     UDB_MLT_G_ENTRY .BLKW   3       ; Entry has a 48 bit logical address
                     0006  1257 $DEF     UDB_MLT_C_LENGTH                ; Size of each logical address
0000003C             0006  1258 UDB_MLT_K_LENGTH = UDB_MLT_C_LENGTH * MAX_C_MLT ; Size of multicast list
                     0006  1259
                     0006  1260         $DEFEND UDB                     ; End of UDBB block descriptors
                     0000  1261 ;
                     0000  1262 ; Define the Transmit Ring Entry
                     0000  1263 ;
                     0000  1264         $DEFINI XMT     GLOBAL          ; Start of Transmit Ring Entry format
                     0000  1265
                     0000  1266 $DEF     XMT_W_SLEN      .BLKW   1       ; Segment length
                     0002  1267 $DEF     XMT_W_SEGBL     .BLKW   1       ; Segment address (low 16 bits)
                     0004  1268 $DEF     XMT_B_SEGBH     .BLKB   1       ; Segment address (hi 2 bits)
                     0005  1269 $DEF     XMT_B_FLAGS     .BLKB   1       ; Transmit status flags
                     0006  1270 $DEF     XMT_W_TDRERR    .BLKW   1       ; Time domain reflectometry value
                     0008  1271                                          ;& and transmit errors
                     0008  1272 $DEF     XMT_W_RID       .BLKW   1       ; Request ID
                     000A  1273 $DEF     XMT_C_LENGTH                    ; Size of transmit buffer ring entry
00000028             000A  1274 XMT_K_LENGTH = XMT_C_LENGTH * MAX_C_XMT ; Size of xmit ring
                     000A  1275
                     000A  1276         _VIELD  XMT_FLG,0,<-            ; Define flag bits
                     000A  1277                 <ENP,,M>,-              ; End of packet
                     000A  1278                 <STP,,M>,-              ; Start of packet
                     000A  1279                 <DEF,,M>,-              ; Transmit Deferred
                     000A  1280                 <ONE,,M>,-              ; One collision encountered
                     000A  1281                 <MORE,,M>,-             ; More than one retry required
                     000A  1282                 <MTCH,,M>,-             ; Address Match on xmit request
                     000A  1283                 <ERRS,,M>,-             ; Error summary
                     000A  1284                 <OWN,,M>,-              ; If 0 then owned by driver
```

```
                     000A  1285                >
                     000A  1286
                     000A  1287          _VIELD  XMT_TERR,0,<-            ; Define reflectometry and errors
                     000A  1288                  <TDR,10,M>,-            ; Time domain reflectometry value
                     000A  1289                  <RTRY,,M>,-             ; Failed after 16 retries
                     000A  1290                  <LCAR,,M>,-             ; Loss of carrier
                     000A  1291                  <LCOL,,M>,-             ; Late collision
                     000A  1292                  <UFLO,,M>,-             ; Underflow error (DELUA Only)
                     000A  1293                  <UBTO,,M>,-             ; UNIBUS timeout
                     000A  1294                  <BUFL,,M>,-             ; Buffer length error
                     000A  1295                  >
                     000A  1296
                     000A  1297          $DEFEND XMT                     ; End of Transmit Ring Entry
                     0000  1298 ;
                     0000  1299 ; Define the Receive Ring Entry
                     0000  1300 ;
                     0000  1301          $DEFINI RCV     GLOBAL          ; Start of Receive Ring Entry format
                     0000  1302
                     0000  1303 $DEF     RCV_W_SLEN      .BLKW   1       ; Segment length
                     0002  1304 $DEF     RCV_W_SEGBL     .BLKW   1       ; Segment address (low 16 bits)
                     0004  1305 $DEF     RCV_B_SEGBH     .BLKB   1       ; Segment address (hi 2 bits)
                     0005  1306 $DEF     RCV_B_FLAGS     .BLKB   1       ; Receive status flags
                     0006  1307 $DEF     RCV_W_LENERR    .BLKW   1       ; Message length and receive errors
                     0008  1308 $DEF     RCV_W_RID       .BLKW   1       ; Request ID
                     000A  1309 $DEF     RCV_C_LENGTH                    ; Size of receive buffer ring entry
           00000050  000A  1310 RCV_K_LENGTH = RCV_C_LENGTH * MAX_C_RCV ; Size of receive ring
                     000A  1311
                     000A  1312          _VIELD  RCV_FLG,0,<-            ; Define flag bits
                     000A  1313                  <ENP,,M>,-              ; End of packet
                     000A  1314                  <STP,,M>,-              ; Start of packet
                     000A  1315                  <,1>,-                  ; RESERVED
                     000A  1316                  <CRC,,M>,-              ; CRC error
                     000A  1317                  <OFLO,,M>,-             ; Message overflow
                     000A  1318                  <FRAM,,M>,-             ; Framing error
                     000A  1319                  <ERRS,,M>,-             ; Error summary
                     000A  1320                  <OWN,,M>,-              ; If 0 then owned by driver
                     000A  1321                  >
                     000A  1322
                     000A  1323          _VIELD  RCV_LERR,0,<-           ; Define MSG length and error bits
                     000A  1324                  <MLEN,12,M>,-           ; Message length
                     000A  1325                  <OVRN>,-                ; Overrun error (DELUA only)
                     000A  1326                  <NCHN,,M>,-             ; No Data Chaining
                     000A  1327                  <UBTO,,M>,-             ; UNIBUS timeout
                     000A  1328                  <BUFL,,M>,-             ; Buffer length error
                     000A  1329                  >
                     000A  1330
                     000A  1331          $DEFEND RCV                     ; End of Receive Ring Entry
                     0000  1332
                     0000  1333 ;
                     0000  1334 ; Transmit Buffer Header Format
                     0000  1335 ;
                     0000  1336          $DEFINI XBUF                    ; Define transmit buffer header
                     0000  1337
                     0000  1338 $DEF     XBUF_G_DEST     .BLKW   3       ; Destination address
                     0006  1339 $DEF     XBUF_G_SRC                      ; Source address (overlays UCB)
                     0006  1340 $DEF     XBUF_W_UNUSED   .BLKW   1       ;   sometimes overwritten with DEST
                     0008  1341 $DEF     XBUF_L_UCB      .BLKL   1       ; UCB address
```

```
000C  1342 $DEF     XBUF_W_TYPE      .BLKW   1       ; Protocol type
000E  1343 $DEF     XBUF_T_DATA                      ; Start of xmit data
000E  1344 $DEF     XBUF_C_HEADER                    ; Size of buffer header
000E  1345
000E  1346 $DEF     XBUF_W_SIZE      .BLKW   1       ; Size of buffer (only if padding)
0010  1347
0010  1348          $DEFEND XBUF                     ; End of transmit buffer header
0000  1349
0000  1350 ;
0000  1351 ; Block header for non-DECnet xmit buffers
0000  1352 ;
0000  1353          $DEFINI BLK                      ; Define a standard block header
0000  1354
0000  1355 $DEF     BLK_L_LINK       .BLKL   2       ; Forward and backward queue links
0008  1356 $DEF     BLK_W_SIZE       .BLKW   1       ; Block size
000A  1357 $DEF     BLK_B_TYPE       .BLKB   1       ; Block type
000B  1358 $DEF     BLK_B_SPARE      .BLKB   1       ; SPARE byte
000C  1359 $DEF     BLK_T_DATA                       ; Start of data
000C  1360 $DEF     BLK_C_HEADER                     ; Size of buffer header
000C  1361
000C  1362          $DEFEND BLK
```

```
                        0000  1364 ;
                        0000  1365 ; Define the DEUNA/DELUA Controller Data Block (CDB) fields
                        0000  1366 ;
                        0000  1367
                        0000  1368          $DEFINI CDB     GLOBAL          ; Start of CDB definitions
                        0000  1369
                        0000  1370 $DEF     CDB_L_FQFL      .BLKL   1       ; Fork queue forward link
                        0004  1371 $DEF     CDB_L_FQBL      .BLKL   1       ; Fork queue backward link
                        0008  1372 $DEF     CDB_W_SIZE      .BLKW   1       ; Size of CDB
                        000A  1373 $DEF     CDB_B_TYPE      .BLKB   1       ; Type of structure
                        000B  1374 $DEF     CDB_B_FIPL      .BLKB   1       ; Fork IPL
                        000C  1375 $DEF     CDB_L_FPC       .BLKL   1       ; Fork PC
                        0010  1376 $DEF     CDB_L_PCSR0                     ; Port CSR0 and CSR1 contents
                        0010  1377 $DEF     CDB_L_FR3       .BLKL   1       ; Fork R3
                        0014  1378 $DEF     CDB_L_FR4       .BLKL   1       ; Fork R4
                        0018  1379 $DEF     CDB_B_NEXTXMT   .BLKB   1       ; Next entry in XMT ring
                        0019  1380 $DEF     CDB_B_NEXTRCV   .BLKB   1       ; Next entry in RCV ring
                        001A  1381 ASSUME   MAX_C_XMT LE 8
                        001A  1382 ASSUME   MAX_C_RCV LE 8
                        001A  1383 $DEF     CDB_B_RCVMAP    .BLKB   1       ; RCV map slot in use flags
                        001B  1384 $DEF     CDB_B_XMTMAP    .BLKB   1       ; XMT map slot in use flags
                        001C  1385 $DEF     CDB_L_RCVMAP    .BLKL   MAX_C_RCV ; RCV mapping vector
                        003C  1386 $DEF     CDB_L_XMTMAP    .BLKL   MAX_C_XMT ; XMT mapping vector
                        004C  1387 $DEF     CDB_L_XRADDR    .BLKL   MAX_C_XMT ; XMT ring entry address vector
                        005C  1388 $DEF     CDB_L_RRADDR    .BLKL   MAX_C_RCV ; RCV ring entry address vector
                        007C  1389
                        007C  1390 $DEF     CDB_Q_QUEUES                    ; Start of CDB queues
                        007C  1391 $DEF     CDB_Q_XMTREQ    .BLKQ   1       ; Transmit request queue
= 00000001              0084  1392 CDB_C_ABORTS = <.-CDB_Q_QUEUES>/8        ; Number of Queues to abort requests
                        0084  1393 $DEF     CDB_Q_INPUT     .BLKQ   1       ; Input process queue
                        008C  1394 $DEF     CDB_Q_DONE      .BLKQ   1       ; IRP waiting DNI to set
                        0094  1395 $DEF     CDB_Q_XMTPND    .BLKQ   1       ; Transmit pending queue
                        009C  1396 $DEF     CDB_Q_RCVBUF    .BLKQ   1       ; Receive buffer queue
                        00A4  1397 $DEF     CDB_Q_RCVPND    .BLKQ   1       ; Receive pending queue
                        00AC  1398 $DEF     CDB_Q_POST      .BLKQ   1       ; Post process queue
= 00000007              00B4  1399 CDB_C_QUEUES = <.-CDB_Q_QUEUES>/8        ; Number of Queue Heads
                        00B4  1400
                        00B4  1401 $DEF     CDB_B_LASTXMT   .BLKB   1       ; Last entry done in XMT ring
                        00B5  1402 $DEF     CDB_B_LASTRCV   .BLKB   1       ; Last entry done in RCV ring
                        00B6  1403 $DEF     CDB_B_RCVCNT    .BLKB   1       ; Count of receives given to DEUNA/DELUA
                        00B7  1404 $DEF     CDB_B_XMTCNT    .BLKB   1       ; Count of xmits given to DEUNA/DELUA
                        00B8  1405 $DEF     CDB_W_BSZ       .BLKW   1       ; Device buffer size
                        00BA  1406 $DEF     CDB_W_QUOTA     .BLKW   1       ; SYSTEM buffer quota
                        00BC  1407
                        00BC  1408 $DEF     CDB_L_PCBBMAP   .BLKL   1       ; PCBB UNIBUS map register number/count
                        00C0  1409 $DEF     CDB_L_PCBBUA    .BLKL   1       ; PCBB UNIBUS address
                        00C4  1410 $DEF     CDB_L_DEVDEPEND .BLKL   1       ; Device dependent longword
                        00C8  1411 $DEF     CDB_L_UCB0      .BLKL   1       ; Address of UCB #0
                        00CC  1412 $DEF     CDB_B_SPARE     .BLKB   1       ; SPARE BYTE
                        00CD  1413 $DEF     CDB_B_DIAG1     .BLKB   1       ; Diagnostic info byte
                        00CE  1414 $DEF     CDB_W_DIAG2     .BLKW   1       ; Second word of diagnostic info
                        00D0  1415
                        00D0  1416 $DEF     CDB_W_UFDCTR    .BLKW   1       ; No protocol type counter
                        00D2  1417 $DEF     CDB_W_SBUCTR    .BLKW   1       ; System buffer unavailable
                        00D4  1418 $DEF     CDB_W_UBUCTR    .BLKW   1       ; No buffer available on all PTs
                        00D6  1419 ;
                        00D6  1420 ; Define the UNIBUS mapped portion of the CDB (DEUNA/DELUA data structures)
```

```
                        00D6  1421 ;
                        00D6  1422 $DEF     CDB_G_MAPPED                        ; Start of UNIBUS mapped portion of CDB
                        00D6  1423 $DEF     CDB_G_PCBB          .BLKB   PBB_K_LENGTH    ; PCBB
                        00DE  1424 $DEF     CDB_G_XRUDB         .BLKB   UDB_BUF_K_LENGTH ; XMIT/RECV buffer descriptor
                        00EA  1425 $DEF     CDB_G_CUDB          .BLKB   UDB_CTR_K_LENGTH ; Counter buffer
                        012A  1426 $DEF     CDB_G_MUDB          .BLKB   UDB_MLT_K_LENGTH ; Multicast address buffer
                        0166  1427 $DEF     CDB_G_SCRUDB        .BLKB   8           ; SCRATCH UDBB
                        016E  1428 $DEF     CDB_G_SIDUDB        .BLKB   UDB_SID_C_LENGTH ; System ID UDBB
                        01AA  1429 $DEF     CDB_G_XRING         .BLKB   XMT_K_LENGTH    ; Xmit ring buffer
                        C1D2  1430 $DEF     CDB_G_RRING         .BLKB   RCV_K_LENGTH    ; Recv ring buffer
0000014C                0222  1431 CDB_C_MAPPED = .-CDB_G_MAPPED                ; Size of UNIBUS mapped portion of CDB
                        0222  1432 $DEF     CDB_C_ZERO                          ; Size of CDB to zero (everything from
                        0222  1433                                              ;  the beginning to this point).
                        0222  1434 $DEF     CDB_B_TIM_XMT       .BLKB   1       ; Transmit timer cell
                        0223  1435 $DEF     CDB_B_TIM_DNI       .BLKB   1       ; DNI timer cell
                        0224  1436 $DEF     CDB_L_PRMUSER       .BLKL   1       ; Promiscuous user's UCB address
                        0228  1437 $DEF     CDB_L_TQE           .BLKB   TQE$C_LENGTH ; Timer queue element
                        0258  1438 $DEF     CDB_W_MODE          .BLKW   1       ; DEUNA/DELUA hardware mode
                        025A  1439 $DEF     CDB_B_STS           .BLKB   1       ; Controller status
                        025B  1440 $DEF     CDB_B_PAD           .BLKB   1       ; Padding mode
                        025C  1441 $DEF     CDB_B_DCH           .BLKB   1       ; Data chaining
                        025D  1442 $DEF     CDB_B_PRM           .BLKB   1       ; Promiscuous mode
                        025E  1443 $DEF     CDB_B_MLT           .BLKB   1       ; Multicast (all) address state
                        025F  1444
                        025F  1445 $DEF     CDB_B_SETPRM                        ; Start of settable parameters
                        025F  1446 $DEF     CDB_B_CRC           .BLKB   1       ; CRC generation mode
                        0260  1447 $DEF     CDB_B_CON           .BLKB   1       ; Controller mode
                        0261  1448 $DEF     CDB_B_EKO           .BLKB   1       ; Echo mode (DEUNA Only)
                        0262  1449 $DEF     CDB_B_ILP           .BLKB   1       ; Internal Loopback mode (DELUA Only)
00000004                0263  1450 CDB_C_SETPRM = .-CDB_B_SETPRM                 ; Size of settable parameter list
                        0263  1451
                        0263  1452 $DEF     CDB_G_PHA           .BLKW   3       ; User defined physical address
                        0269  1453 $DEF     CDB_G_HWA           .BLKW   3       ; Hardware physical address
                        026F  1454 $DEF     CDB_G_PHYADR        .BLKW   3       ; The current hardware address
                        0275  1455 $DEF     CDB_B_UNTCNT        .BLKB   1       ; Number of active units (UCBs)
                        0276  1456 $DEF     CDB_B_MULTI         .BLKB   1       ; Number of entries in Multicast list
                        0277  1457 $DEF     CDB_B_MLTTBL        .BLKB   1       ; Number of entries in MLTTBL
                        0278  1458 $DEF     CDB_G_MLTTBL        .BLKW   3*MAX_C_MLT ; Multicast address generation tbl
                        02B4  1459 $DEF     CDB_L_ASSPKT        .BLKL   1       ; Address of ASSPKT routine to be used
                        02B8  1460 $DEF     CDB_B_AQUOTA        .BLKB   1       ; Additional QUOTA available
                        02B9  1461 $DEF     CDB_B_MQUOTA        .BLKB   1       ; Maximum extra QUOTA allowed
                        02BA  1462
                        02BA  1463 $DEF     CDB_C_LENGTH                        ; Size of DEUNA/DELUA CDB
                        02BA  1464
                        02BA  1465          _VIELD  CDB_STS,0,<-                ; CDB status bit for CDB_B_STS
                        02BA  1466                  <INITED,,M>,-               ; Inited
                        02BA  1467                  <RUN,,M>,-                  ; Device is running
                        02BA  1468                  <FORK_PEND,,M>,-            ; Fork process is pending
                        02BA  1469                  <FUNC,,M>,-                 ; Command function is NOT pending
                        02BA  1470                  <TIMER,,M>,-                ; Timer is active
                        02BA  1471                  <RCBI,,M>,-                 ; Receive buffer unavailable error
                        02BA  1472                  <ERR,,M>,-                  ; Fatal error has occurred
                        02BA  1473                  <POWER,,M>,-                ; Power Failure has occurred
                        02BA  1474                  >
                        02BA  1475
                        02BA  1476          $DEFEND CDB                         ; End of DEUNA/DELUA CDB definitions
```

```
0000  1478 ;
0000  1479 ; P2 buffer header definition
0000  1480 ;
0000  1481          $DEFINI P2B
0000  1482
0000  1483 $DEF     P2B_L_POINTER   .BLKL   1       ; Pointer to start of data
0004  1484 $DEF     P2B_L_BUFFER    .BLKL   1       ; Address of user's data buffer
0008  1485 $DEF     P2B_W_SIZE      .BLKW   1       ; Size of P2 buffer
000A  1486 $DEF     P2B_B_TYPE      .BLKB   1       ; Type of structure
000B  1487 $DEF     P2B_B_SPARE     .BLKB   1       ; Spare byte
000C  1488 $DEF     P2B_C_LENGTH                    ; Size of P2 buffer header
000C  1489 $DEF     P2B_T_DATA                      ; Start of data
000C  1490
000C  1491          $DEFEND P2B
0000  1492 ;
0000  1493 ; Diagnostics buffer definition
0000  1494 ;
0000  1495          $DEFINI DIAG
0000  1496 ;
0000  1497 ; Driver independent portion of diagnostics buffer
0000  1498 ;
0000  1499 $DEF     DIAG_L_DATA     .BLKL   1       ; Pointer to start of data
0004  1500 $DEF     DIAG_L_BUFFER   .BLKL   1       ; User buffer address
0008  1501 $DEF     DIAG_W_SIZE     .BLKW   1       ; Size of structure
000A  1502 $DEF     DIAG_B_TYPE     .BLKB   1       ; Type of structure
000B  1503 $DEF     DIAG_B_SPARE    .BLKB   1       ; Spare byte
000C  1504 $DEF     DIAG_T_DATA                     ; Start of data
000C  1505 $DEF     DIAG_Q_START    .BLKQ   1       ; Start time for QIO
0014  1506 $DEF     DIAG_Q_FINISH   .BLKQ   1       ; Finish time for QIO
001C  1507 $DEF     DIAG_L_ERRS     .BLKL   1       ; Number of device errors
0020  1508 $DEF     DIAG_L_EXTRA    .BLKL   1       ; Number of longwords that follow
0024  1509 ;
0024  1510 ; Driver dependent portion of diagnostics buffer
0024  1511 ;
0024  1512 $DEF     DIAG_L_DEPEND
0024  1513 $DEF     DIAG_Q_PCSR     .BLKQ   1       ; Last port CSR contents
002C  1514 $DEF     DIAG_Q_PCBB     .BLKQ   1       ; Last port command block contents
0034  1515 $DEF     DIAG_W_ERR      .BLKW   1       ; Ring entry error summary
0036  1516 $DEF     DIAG_G_HWA      .BLKW   3       ; Hardware physical address
003C  1517 $DEF     DIAG_W_ERR2     .BLKW   1       ; Extra ring entry error info
003E  1518 ;
003E  1519 ; The following is valid only on read (receive) QIOs
003E  1520 ;
003E  1521 $DEF     DIAG_T_RDATA                    ; Start of receive data
003E  1522 $DEF     DIAG_G_DEST     .BLKW   3       ; Destination address
0044  1523 $DEF     DIAG_G_SRC      .BLKW   3       ; Source address
004A  1524 $DEF     DIAG_W_TYPE     .BLKW   1       ; Protocol type
004C  1525 $DEF     DIAG_C_LENGTH                   ; Start of data
0000000A  004C  1526 DIAG_C_EXTRA = .-DIAG_L_DEPEND/4
004C  1527
004C  1528          $DEFEND DIAG
0000  1529 ;
0000  1530 ; Receive buffer header definition
0000  1531 ;
0000  1532          $DEFINI RHDR
0000  1533
0000  1534 $DEF     RHDR_L_DATA     .BLKL   1       ; Pointer to start of data
```

```
                     0004  1535 $DEF     RHDR_L_BUFFER    .BLKL   1        ; User buffer address
                     0008  1536 $DEF     RHDR_W_SIZE      .BLKW   1        ; Size of structure
                     000A  1537 $DEF     RHDR_B_TYPE      .BLKB   1        ; Type of structure
                     000B  1538 $DEF     RHDR_B_SPARE     .BLKB   1        ; Spare byte
                     000C  1539 $DEF     RHDR_T_DATA                       ; Start of data
                     000C  1540 $DEF     RHDR_G_DEST      .BLKW   3        ; Destination address
                     0012  1541 $DEF     RHDR_G_SRC       .BLKW   3        ; Source address
                     0018  1542 $DEF     RHDR_W_TYPE      .BLKW   1        ; Protocol type
                     001A  1543 $DEF     RHDR_C_LENGTH                     ; Start of data
0000000E             001A  1544 RHDR_C_DATA = .-RHDR_T_DATA
                     001A  1545
                     001A  1546          $DEFEND RHDR
                     0000  1547
                     0000  1548 ;
                     0000  1549 ; Shareable protocol type data structure
                     0000  1550 ;
                     0000  1551          $DEFINI SHR      GLOBAL
                     0000  1552 $DEF     SHR_L_QFL        .BLKL   1        ; Forward link pointer
                     0004  1553 $DEF     SHR_L_QBL        .BLKL   1        ; Backward link pointer
                     0008  1554 $DEF     SHR_W_SIZE       .BLKW   1        ; Size of structure
                     000A  1555 $DEF     SHR_B_TYPE       .BLKB   1        ; Type of structure
                     000B  1556 $DEF     SHR_B_STS        .BLKB   1        ; SHR status
                     000C  1557 $DEF     SHR_L_PID        .BLKL   1        ; User's PID
                     0010  1558 $DEF     SHR_W_CHAN       .BLKW   1        ; User's channel
                     0012  1559 $DEF     SHR_G_DEST       .BLKW   3        ; Destination address
                     0018  1560 $DEF     SHR_Q_QUEUES                      ; Start of queues
                     0018  1561 $DEF     SHR_Q_RCVMSG     .BLKQ   1        ; Received messages waiting for IRPs
                     0020  1562 $DEF     SHR_Q_RCVREQ     .BLKQ   1        ; Receive IRPs waiting for messages
00000002             0028  1563 SHR_C_QUEUES = <.-SHR_Q_QUEUES>/8          ; Number of queues
                     0028  1564 $DEF     SHR_W_QUOTA      .BLKW   1        ; User's shared quota
                     002A  1565 $DEF     SHR_C_LENGTH                      ; Size of data structure
                     002A  1566
                     002A  1567          _VIELD  SHR_STS,0,<-              ; SHR status bits
                     002A  1568                  <INITED,,M>,-             ; Protocol type is inited
                     002A  1569                  >
                     002A  1570
                     002A  1571          $DEFEND SHR
```

```
0000  1573          .SBTTL  DEUNA/DELUA device driver diagnostic function codes
0000  1574
0000  1575 $EQU     XE$C_DG_LOADMEM          1       ; Write DEUNA/DELUA memory
0000  1576 $EQU     XE$C_DG_READMEM          2       ; Read DEUNA/DELUA memory
0000  1577 $EQU     XE$C_DG_UPADDR           3       ; Load and start u-proc address
0000  1578 $EQU     XE$C_DG_WRITEID          4       ; Write SYSTEM ID parameters
0000  1579 $EQU     XE$C_DG_READID           5       ; Read SYSTEM ID parameters
0000  1580 $EQU     XE$C_DG_STOP             6       ; Stop transmit and receive process
0000  1581 $EQU     XE$C_DG_STEST            7       ; Perform a self test
0000  1582 $EQU     XE$C_DG_READST           8       ; Read port status
0000  1583 $EQU     XE$C_DG_ESTEST           9       ; Perform an extended self test (DELUA)
0000  1584 $EQU     XE$C_DG_HALT            10       ; Halt port (DELUA Only)
```

```
                        1586          .SBTTL  Standard tables
                        1587 ;
                        1588 ; Driver prologue table
                        1589 ;
0000                    1590          DPTAB   -                                       ;
0000                    1591                  END=XE_END,-                            ; END OF DRIVER
0000                    1592                  ADAPTER=UBA,-                           ; ADAPTER TYPE
0000                    1593                  UCBSIZE=UCB$L_XE_LENGTH,-               ; SIZE OF UCB
0000                    1594                  NAME=XEDRIVER                           ; DRIVER NAME
0038                    1595
0038                    1596          DPT_STORE INIT                                  ; START OF CONTROLLER INIT
0038                    1597          DPT_STORE UCB,UCB$B_FIPL,B,IPL$_XE_FIPL         ; FORK IPL
003C                    1598          DPT_STORE UCB,UCB$B_DIPL,B,IPL$_XE_DIPL         ; DEVICE IPL
0040                    1599          DPT_STORE ORB,ORB$B_FLAGS,B,-                   ; Protection block flags
0040                    1600                    <ORB$M_PROT_16>                       ; SOGW protection word
0044                    1601          DPT_STORE ORB,ORB$W_PROT,W,0                    ; default protection
0049                    1602          DPT_STORE ORB,ORB$L_OWNER,L,0                   ; no owner as yet
0050                    1603          DPT_STORE UCB,UCB$L_DEVCHAR,L,- ; DEVICE CHARACTERISTICS
0050                    1604                    <DEV$M_SHR!-
0050                    1605                     DEV$M_AVL!-
0050                    1606                     DEV$M_NET!-
0050                    1607                     DEV$M_IDV!-
0050                    1608                     DEV$M_ODV>
0057                    1609
0057                    1610          DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_SCOM         ; Device class
005B                    1611          DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_DEUNA         ; Device type
005F                    1612          DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,512             ; Default buffer size
0064                    1613          DPT_STORE UCB,UCB$W_STS,W,<UCB$M_ONLINE!UCB$M_TEMPLATE>
0069                    1614          DPT_STORE UCB,UCB$G_XE_PHA,L,-1                 ; No default physical address
0070                    1615          DPT_STORE UCB,UCB$G_XE_PHA+4,W,-1               ;       ...
0075                    1616 ;
0075                    1617 ; Store defaults for all parameters
0075                    1618 ;
0075                    1619          DPT_STORE UCB,UCB$W_XE_BSZ,W,1500               ; Default device buffer size
007A                    1620          DPT_STORE UCB,UCB$B_XE_BFN,B,1                  ; Default user buffer number
007E                    1621          DPT_STORE UCB,UCB$W_XE_HBQ,W,INIT_C_QUOTA ; Hardware Buffer Quota
0083                    1622          DPT_STORE UCB,UCB$B_XE_PRO,B,NMA$C_LINPR_NI ; "NI" is the protocol mode
0087                    1623          DPT_STORE UCB,UCB$B_XE_PRM,B,NMA$C_STATE_OFF ; Promiscuous mode is OFF
008B                    1624          DPT_STORE UCB,UCB$B_XE_MLT,B,NMA$C_STATE_OFF ; All multicasts is OFF
008F                    1625          DPT_STORE UCB,UCB$B_XE_DCH,B,NMA$C_STATE_OFF ; Data chaining is OFF
0093                    1626          DPT_STORE UCB,UCB$B_XE_CRC,B,NMA$C_STATE_ON ; CRC generation is ON
0097                    1627          DPT_STORE UCB,UCB$B_XE_PAD,B,NMA$C_STATE_ON ; Padding is ON
009B                    1628          DPT_STORE UCB,UCB$B_XE_CON,B,NMA$C_LINCN_NOR ; Controler mode is NORMAL
009F                    1629          DPT_STORE UCB,UCB$B_XE_EKO,B,NMA$C_STATE_OFF ; No ECHO is default (DEUNA)
00A3                    1630          DPT_STORE UCB,UCB$B_XE_ACC,B,NMA$C_ACC_EXC ; Exclusive mode is default
00A7                    1631          DPT_STORE UCB,UCB$B_XE_ILP,B,NMA$C_STATE_OFF ; Internal Loopback is OFF (DEL
00AB                    1632
00AB                    1633          DPT_STORE REINIT
00AB                    1634
00AB                    1635          DPT_STORE DDB,DDB$L_DDT,D,XE$DDT                ; DDT ADDRESS
00B0                    1636          DPT_STORE CRB,CRB$L_INTD+4,D,DONE_INTR  ; Completion interrupt
00B5                    1637          DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,D,CONTROL_INIT; CONTROLLER INIT ADDRE
00BA                    1638          DPT_STORE CRB,CRB$L_INTD+VEC$L_UNITINIT,D,UNIT_INIT; UNIT INIT
00BF                    1639          DPT_STORE CRB,CRB$L_INTD+VEC$L_START,D,FFI_INIT ; FFI INIT
00C4                    1640          DPT_STORE END
0000                    1641
00000000                1642          .PSECT  $$$115_DRIVER,LONG
```

```
0000  1643 ;
0000  1644 ; LOCAL STORAGE
0000  1645 ;
0000  1646 ; Driver dispatch table
0000  1647 ;
0000  1648         DDTAB   DEVNAM=XE,-                     ; DRIVER DISPATCH TABLE
0000  1649                 START=STARTIO,-                 ; Start I/O operation
0000  1650                 FUNCTB=XE_FUNCTABLE,-           ; Function decision table address
0000  1651                 CANCEL=CANCEL,-                 ; CANCEL I/O entry point
0000  1652                 REGDMP=REG_DUMP,-               ; Register dump entry point
0000  1653                 DIAGBF=<DIAG_C_LENGTH>,-        ; Diagnostic buffer size
0000  1654                 CLONEDUCB=CLONED_UCB,-          ; Cloned UCB initialization
0000  1655                 ALTSTART=ALT_START              ; Alternate start I/O entry point
0038  1656 ;
0038  1657 ; Function decision table
0038  1658 ;
0038  1659 XE_FUNCTABLE:                                  ;
0038  1660         FUNCTAB ,-                              ; Legal Function
0038  1661         <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
0038  1662         READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR,DIAGNOSE>
0040  1663         FUNCTAB ,-                              ; Buffered Functions
0040  1664         <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
0040  1665         READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR,DIAGNOSE>
0048  1666         FUNCTAB XMT_FDT,<WRITELBLK,WRITEPBLK,WRITEVBLK> ;
0054  1667         FUNCTAB RCV_FDT,<READLBLK,READPBLK,READVBLK> ;
0060  1668         FUNCTAB SETMODE_FDT,<SETMODE,SETCHAR> ;
006C  1669         FUNCTAB SENSEMODE_FDT,<SENSEMODE,SENSECHAR> ;
0078  1670         FUNCTAB DIAGNOSE_FDT,<DIAGNOSE> ; Diagnostic functions
0084  1671
```

```
                        .SBTTL  Local driver storage
                ;
                ; Initial setting for DEUNA/DELUA mode
                ;
2800    0084    DEF_MODE::      .WORD   PB_MOD_M_ECT!-  ; Enable the H4000 collision checks
        0086                            PB_MOD_M_DRDC   ; Disable Data Chaining
        0086
                ;
                ; P2 Buffer verification tables
                ;
        0086
        0086            $DEFINI PARAM
        0000
        0000    $DEF    PRM_W_TYPE      .BLKW   1       ; Parameter type
        0002
        0002            _VIELD  PRM_TYP,0,<-            ; Parameter type field
        0002                    <CODE,12,M>,-           ; Parameter type code
        0002                    <STRING,1,M>,-          ; Parameter is a string
        0002                    >
        0002
        0002    $DEF    PRM_B_FLAG      .BLKB   1       ; Parameter flags
        0003
        0003            _VIELD  PRM_FLG,0,<-            ; Parameter flag bits
        0003                    <MIN,1,M>,-             ; Parameter minimum value present
        0003                    <MAX,1,M>,-             ; Parameter maximum value present
        0003                    <INVALID,1,M>,-         ; Parameter invalid value is present
        0003                    <CDB,1,M>,-             ; Offset is in CDB data base
        0003                    <CHECK,1,M>,-           ; Check values with current
        0003                    >
        0003
        0003    $DEF    PRM_W_OFF       .BLKW   1       ; Parameter offset in structure
        0005
        0005            _VIELD  PRM_OFF,0,<-            ; Offset word fields
        0005                    <VALUE,10,M>,-          ; Offset value
        0005                    <WIDTH,6,M>,-           ; Size of field in structure
        0005                    >
        0005
        0005            $DEFEND PARAM
        0086
                ;
                ; Define Line parameters
                ;
00000000    0086    LINE_PRM_BUFSIZ=0                       ; Line parameter buffer size
        0086    LINE_PARAM_WO:                          ; "Write-Only" line parameters
        0086
        0086            PARAM   NMA$C_PCLI_HBQ,-        ; Hardware Buffer Quota
        0086                    OFFSET=UCB$W_XE_HBQ,-
        0086                    WIDTH=W,MAX=16384,-
        0086                    INVALID=UCB$M_XE_INITED
        008F
        008F    LINE_PARAM:                             ; Start of line parameters
        008F
        008F            PARAM   NMA$C_PCLI_ACC,-        ; Access mode for protocol type
        008F                    OFFSET=UCB$B_XE_ACC,-
        008F                    WIDTH=B,-
        008F                    MIN=NMA$C_ACC_SHR,-
        008F                    MAX=NMA$C_ACC_EXC
```

```
0098  1730
0098  1731          PARAM   NMA$C_PCLI_PRO,-         ; Protocol selection mode
0098  1732                  OFFSET=UCB$B_XE_PRO,-
0098  1733                  WIDTH=B,-
0098  1734                  MIN=NMA$C_LINPR_POI,-    ;% Accept either point or NI
0098  1735                  MAX=NMA$C_LINPR_NI
00A1  1736
00A1  1737          PARAM   NMA$C_PCLI_BUS,-         ; Buffer size
00A1  1738                  OFFSET=UCB$W_DEVBUFSIZ,-
00A1  1739                  WIDTH=W,-
00A1  1740                  MIN=MIN_PKT_SIZE,-       ; User buffer LIMITS
00A1  1741                  MAX=MAX_PKT_SIZE,-
00A1  1742                  INVALID=UCB$M_XE_INITED
00AC  1743
00AC  1744          PARAM   NMA$C_PCLI_BFN,-         ; Buffer number
00AC  1745                  OFFSET=UCB$B_XE_BFN,-
00AC  1746                  WIDTH=B,-
00AC  1747                  MIN=0,MAX=255,-
00AC  1748                  INVALID=UCB$M_XE_INITED
00B7  1749
00B7  1750          PARAM   NMA$C_PCLI_PHA,-         ; Physical NI address
00B7  1751                  OFFSET=UCB$G_XE_PHA,-
00B7  1752                  STRING=YES,-
00B7  1753                  SIZE=<2+6>,-
00B7  1754                  INVALID=UCB$M_XE_INITED
00BE  1755
00BE  1756          PARAM   NMA$C_PCLI_PRM,-         ; Promiscuous mode state
00BE  1757                  OFFSET=UCB$B_XE_PRM,-
00BE  1758                  WIDTH=B,-
00BE  1759                  MAX=NMA$C_STATE_OFF,-
00BE  1760                  INVALID=UCB$M_XE_INITED
00C7  1761
00C7  1762          PARAM   NMA$C_PCLI_MLT,-         ; Accept all multicast addresses
00C7  1763                  OFFSET=UCB$B_XE_MLT,-
00C7  1764                  WIDTH=B,-
00C7  1765                  MAX=NMA$C_STATE_ON,-
00C7  1766                  INVALID=UCB$M_XE_INITED
00D0  1767
00D0  1768          PARAM   NMA$C_PCLI_PAD,-         ; PAD on transmits
00D0  1769                  OFFSET=UCB$B_XE_PAD,-
00D0  1770                  WIDTH=B,-
00D0  1771                  MAX=NMA$C_STATE_OFF
00D7  1772
00D7  1773          PARAM   NMA$C_PCLI_DCH,-         ; Data chain on receives
00D7  1774                  OFFSET=UCB$B_XE_DCH,-
00D7  1775                  WIDTH=B,-
00D7  1776                  MAX=NMA$C_STATE_OFF,-
00D7  1777                  INVALID=UCB$M_XE_INITED
00E0  1778
00E0  1779          PARAM   NMA$C_PCLI_CRC,-         ; Generate CRC on transmits
00E0  1780                  OFFSET=UCB$B_XE_CRC,-
00E0  1781                  WIDTH=B,-
00E0  1782                  MAX=NMA$C_STATE_OFF
00E7  1783
00E7  1784          PARAM   NMA$C_PCLI_CON,-         ; Controller mode
00E7  1785                  OFFSET=UCB$B_XE_CON,-
00E7  1786                  WIDTH=B,-
```

```
00E7  1787                    MAX=NMA$C_LINCN_LOO,-
00E7  1788                    INVALID=UCB$M_XE_INITED
00F0  1789
00F0  1790          PARAM     NMA$C_PCLI_PTY,-          ; Protocol type
00F0  1791                    OFFSET=UCB$W_XE_PROTYP,-
00F0  1792                    WIDTH=W,-
00F0  1793                    INVALID=UCB$M_XE_INITED
00F7  1794
00F7  1795          PARAM     NMA$C_PCLI_MCA,-          ; Multicast address list
00F7  1796                    OFFSET=UCB$G_XE_MULTI,-
00F7  1797                    STRING=YES,-
00F7  1798                    SIZE=2+<6*MAX_C_MLT>      ; Maximum size of list
00FC  1799
00FC  1800          PARAM     NMA$C_PCLI_EKO,-          ; Echo transmitted messages (DEUNA Only)
00FC  1801                    OFFSET=UCB$B_XE_EKO,-
00FC  1802                    WIDTH=B,-
00FC  1803                    MAX=NMA$C_STATE_OFF
0103  1804
0103  1805          PARAM     NMA$C_PCLI_BSZ,-          ; Device buffer size
0103  1806                    OFFSET=UCB$W_XE_BSZ,-
0103  1807                    WIDTH=W,-
0103  1808                    MIN=MIN_PKT_SIZE,-
0103  1809                    MAX=MAX_PKT_SIZE,-
0103  1810                    INVALID=UCB$M_XE_INITED
010E  1811
010E  1812          PARAM     NMA$C_PCLI_DES,-          ; Destination Address for shared
010E  1813                    OFFSET=UCB$G_XE_DES,-     ;  Protocol Type
010E  1814                    STRING=YES,-
010E  1815                    SIZE=<2+6>
0113  1816
0113  1817          PARAM     NMA$C_PCLI_ILP,-          ; Internal Loopback mode (DFLUA Only)
0113  1818                    OFFSET=UCB$B_XE_ILP,-
0113  1819                    WIDTH=B,-
0113  1820                    MAX=NMA$C_STATE_OFF,-
0113  1821                    INVALID=UCB$M_XE_INITED
011C  1822
011C  1823          PARAM                               ; End of table
011E  1824
011E  1825 CIRCUIT_PARAM:                               ; Start of circuit parameter table
011E  1826
011E  1827          PARAM     NMA$C_PCCI_MST,-          ; Maintenance state
011E  1828                    OFFSET=UCB$B_XE_MST,-
011E  1829                    WIDTH=B,-
011E  1830                    MIN=NMA$C_STATE_ON,-
011E  1831                    MAX=NMA$C_STATE_OFF
0127  1832
0127  1833          PARAM                               ; End of table
0129  1834
0129  1835 ;
0129  1836 ; Line/circuit counters
0129  1837 ;
0129  1838 CTRTAB:                                      ; Start of counters
0129  1839          COUNTER        ZER, 16, 2          ; Seconds since last zeroed
012D  1840          COUNTER        DBR, 32, 4          ; Packets received
0131  1841          COUNTER        MBL, 32, 8          ; Multicast packets received
0135  1842          COUNTER        RFL, 16, 12, MAP    ; Packets received in error
0139  1843          COUNTER        BRC, 32, 16         ; Bytes received
```

```
013D  1844          COUNTER         MBY, 32, 20     ; Multicast bytes received
0141  1845          COUNTER         OVR, 16, 24     ; Receives lost - Internal buffer error
0145  1846          COUNTER         LBE, 16, 26     ; Receives lost - Local buffer error
0149  1847          COUNTER         DBS, 32, 28     ; Packets transmitted
014D  1848          COUNTER         MBS, 32, 32     ; Multicast packets transmitted
0151  1849          COUNTER         BSM, 32, 36     ; Packets transmitted - several errors
0155  1850          COUNTER         BS1, 32, 40     ; Packets transmitted - 1 error
0159  1851          COUNTER         BID, 32, 44     ; Packets transmitted - deferred
015D  1852          COUNTER         BSN, 32, 48     ; Bytes transmitted
0161  1853          COUNTER         MSN, 32, 52     ; Multicast bytes transmitted
0165  1854          COUNTER         SFL, 16, 56, MAP ; Transmit packets aborted
0169  1855          COUNTER         CDC, 16, 60     ; Transmit collision check failure
016D  1856
016D  1857 LINE_CTR:
016D  1858          DRVRCTR         UFD, 16, UFDCTR ; Unrecognized frame destination
0171  1859          DRVRCTR         SBU, 16, SBUCTR ; System buffer unavailable
0175  1860          DRVRCTR         UBU, 16, UBUCTR ; User buffer unavailable
0179  1861
0179  1862 CIRC_CTR:
017?  1863          DRVRCTR         DBS, 32, SBLCTR, CIRC ; Blocks sent
017D  1864          DRVRCTR         BSN, 32, SBYCTR, CIRC ; Bytes sent
018?  1865          DRVRCTR         DBR, 32, RBLCTR, CIRC ; Blocks received
018?  1866          DRVRCTR         BRC, 32, RBYCTR, CIRC ; Bytes received
0189  1867          DRVRCTR         MNE, 16, MNECTR, CIRC ; Multicast address not enabled
018D  1868          DRVRCTR         UBU, 16, UBUCTR, CIRC ; User buffer unavailable
0191  1869
0191  1870 ;
0191  1871 ; MOP read counters return table (in order of COUNTERs returned)
0191  1872 ;
0191  1873 MOPCTRTAB:                                  ; Start of MOP counters
0191  1874          MOPCTR          16, 2           ; Seconds since last zeroed
0193  1875          MOPCTR          32, 16          ; Bytes received
0195  1876          MOPCTR          32, 48          ; Bytes transmitted
0197  1877          MOPCTR          32, 4           ; Packets received
0199  1878          MOPCTR          32, 28          ; Packets transmitted
019B  187?          MOPCTR          32, 20          ; Multicast bytes received
019D  1880          MOPCTR          32, 8           ; Multicast packets received
019F  1881          MOPCTR          32, 44          ; Packets transmitted - deferred
01A1  1882          MOPCTR          32, 40          ; Packets transmitted - 1 error
01A3  1883          MOPCTR          16, 36          ;(32)Packets transmitted - several errors
01A5  1884          MOPCTR          16, 58, MAP     ; Transmit packets aborted
01A7  1885          MOPCTR          16, 14, MAP     ; Packets received in error
01A9  1886 ;;       MOPCTR          16, XX          ; Unrecognized frame destination
01A9  1887 ;;       MOPCTR          16, 24          ; Receives lost - Internal buffer error
01A9  1888 ;;       MOPCTR          16, 26          ; Receives lost - Local buffer error
01A9  1889 ;;       MOPCTR          16, XX          ; User buffer unavailable
01A9  1890 ;;&&     MOPCTR          16, 60          ; Transmit collision check failure
01A9  1891          MOPCTR                          ; End of table
01AB  1892 ;
01AB  1893 ; Calculate total size of MOP counter return data buffer
01AB  1894 ;
00000033 01AB  1895          MOP_CTR_SIZE = MOP_CTR_SIZE + 1 + 2 + 8 ; Size of counters + MOP header
00000041 01AB  1896          MOP_CTR_SIZE = MOP_CTR_SIZE + XBUF_C_HEADER ; Size of buffer + NI header
01AB  1897
01AB  1898 ;
01AB  1899 ; BAD PARAMETER RETURN TABLE
01AB  1900 ;
```

```
                     01AB  1901 ; Note that the table is in the REVERSE order from that of the UCB.
                     01AB  1902 ;
                     01AB  1903         ASSUME  UCB$B_XE_CRC EQ UCB$B_XE_CDBPRM
                     01AB  1904         ASSUME  UCB$B_XE_CON EQ UCB$B_XE_CRC+1
                     01AB  1905         ASSUME  UCB$B_XE_EKO EQ UCB$B_XE_CON+1
                     01AB  1906         ASSUME  UCB$B_XE_ILP EQ UCB$B_XE_EKO+1
                     01AB  1907 BAD_PARAM_TBL:
                0B17 01AB  1908         .WORD   NMA$C_PCLI_ILP
                0B1F 01AD  1909         .WORD   NMA$C_PCLI_EKO
                0456 01AF  1910         .WORD   NMA$C_PCLI_CON
                0B1C 01B1  1911         .WORD   NMA$C_PCLI_CRC
                     01B3  1912
                     01B3  1913         ASSUME  UCB$W_XE_BSZ EQ UCB$B_XE_SHRPRM
                     01B3  1914         ASSUME  UCB$B_XE_PRO EQ UCB$W_XE_BSZ+2
                     01B3  1915         ASSUME  UCB$B_XE_PAD EQ UCB$B_XE_PRO+1
                     01B3  1916         ASSUME  UCB$B_XE_DCH EQ UCB$B_XE_PAD+1
                     01B3  1917         ASSUME  UCB$B_XE_PRM EQ UCB$B_XE_DCH+1
                     01B3  1918         ASSUME  UCB$B_XE_MLT EQ UCB$B_XE_PRM+1
                     01B3  1919         ASSUME  UCB$B_XE_CDBPRM EQ UCB$B_XE_MLT+1
                0B19 01B3  1920         .WORD   NMA$C_PCLI_MLT
                0B18 01B5  1921         .WORD   NMA$C_PCLI_PRM
                0B1B 01B7  1922         .WORD   NMA$C_PCLI_DCH
                0B1A 01B9  1923         .WORD   NMA$C_PCLI_PAD
                0458 01BB  1924         .WORD   NMA$C_PCLI_PRO
                0B20 01BD  1925         .WORD   NMA$C_PCLI_BSZ
                0B20 01BF  1926         .WORD   NMA$C_PCLI_BSZ          ; This parameter is a word (not byte)
                     01C1  1927 ;
                     01C1  1928 ; CRC-32 generation table
                     01C1  1929 ;
                     01C1  1930 CRCTABLE:
00000000             01C1  1931         .LONG   0
1DB71064             01C5  1932         .LONG   ^X1DB71064
3B6E20C8             01C9  1933         .LONG   ^X3B6E20C8
26D930AC             01CD  1934         .LONG   ^X26D930AC
76DC4190             01D1  1935         .LONG   ^X76DC4190
6B6B51F4             01D5  1936         .LONG   ^X6B6B51F4
4DB26158             01D9  1937         .LONG   ^X4DB26158
5005713C             01DD  1938         .LONG   ^X5005713C
EDB88320             01E1  1939         .LONG   ^XEDB88320
F00F9344             01E5  1940         .LONG   ^XF00F9344
D6D6A3E8             01E9  1941         .LONG   ^XD6D6A3E8
CB61B38C             01ED  1942         .LONG   ^XCB61B38C
9B64C2B0             01F1  1943         .LONG   ^X9B64C2B0
86D3D2D4             01F5  1944         .LONG   ^X86D3D2D4
A00AE278             01F9  1945         .LONG   ^XA00AE278
BDBDF21C             01FD  1946         .LONG   ^XBDBDF21C
```

```
                0201  1948          .SBTTL  CONTROL_INIT - INITIALIZE DEUNA/DELUA DEVICE
                0201  1949 ;++
                0201  1950 ; CONTROL_INIT - INITIALIZE DEUNA/DELUA DEVICE
                0201  1951 ;
                0201  1952 ; Functional description:
                0201  1953 ;
                0201  1954 ; This routine is entered when driver is loaded, system is booted, or
                0201  1955 ; during powerfail recovery.
                0201  1956 ;
                0201  1957 ; Inputs:
                0201  1958 ;
                0201  1959 ;       R4 = Address of the device CSR
                0201  1960 ;       R5 = Address of the device IDB
                0201  1961 ;       R6 = Address of the device DDB
                0201  1962 ;       R8 = Address of the device CRB
                0201  1963 ;
                0201  1964 ;       IPL = FIPL
                0201  1965 ;
                0201  1966 ; Outputs:
                0201  1967 ;
                0201  1968 ;       R4,R5,R8 are preserved
                0201  1969 ;--
                0201  1970
                0201  1971 CONTROL_INIT::                          ; Initialize the DEUNA/DELUA
     05         0201  1972          RSB                           ; Return to caller
```

```
                     0202  1974          .SBTTL  CLONED_UCB - INITIALIZE THE CLONED UCB
                     0202  1975 ;++
                     0202  1976 ; CLONED_UCB - INITIALIZE THE CLONED UCB
                     0202  1977 ;
                     0202  1978 ; Functional description:
                     0202  1979 ;
                     0202  1980 ; This routine is called by the $ASSIGN system service to allow the driver
                     0202  1981 ; to initialize the cloned UCB. The driver is called with process context.
                     0202  1982 ;
                     0202  1983 ; Inputs:
                     0202  1984 ;
                     0202  1985 ;       R0 = SS$_NORMAL
                     0202  1986 ;       R2 = UCB address of cloned UCB
                     0202  1987 ;       R3 = DDT address
                     0202  1988 ;       R4 = PCB address
                     0202  1989 ;       R5 = UCB address of template UCB
                     0202  1990 ;
                     0202  1991 ;       IPL = ASTDEL
                     0202  1992 ;
                     0202  1993 ; Outputs:
                     0202  1994 ;
                     0202  1995 ;       R0 = SS$_NORMAL
                     0202  1996 ;       R5 = UCB address of cloned UCB
                     0202  1997 ;       All other registers and IPL are preserved.
                     0202  1998 ;--
                     0202  1999
                     0202  2000 CLONED_UCB::                             ; Cloned UCB initialization
55   52   D0        0202  2001          MOVL    R2,R5                   ; Copy UCB address
                     0205  2002                                          ; Continue in unit_initialization
```

```
                        0205  2004          .SBTTL  UNIT_INIT - INITIALIZE THE DEUNA/DELUA UNIT
                        0205  2005 ;++
                        0205  2006 ; UNIT_INIT - INITIALIZE THE DEUNA/DELUA UNIT
                        0205  2007 ;
                        0205  2008 ; Functional description:
                        0205  2009 ;
                        0205  2010 ; This routine is called at system startup, during driver loading and
                        0205  2011 ; during powerfail recovery to initialize the DEUNA/DELUA unit and its UCB.
                        0205  2012 ; The UCB is initialized and if power has failed, the device is forced
                        0205  2013 ; to shutdwon.
                        0205  2014 ;
                        0205  2015 ; Inputs:
                        0205  2016 ;
                        0205  2017 ;       R4 = CSR address
                        0205  2018 ;       R5 = UCB address
                        0205  2019 ;
                        0205  2020 ;       IPL = FIPL
                        0205  2021 ;
                        0205  2022 ; Outputs:
                        0205  2023 ;
                        0205  2024 ;       None.
                        0205  2025 ;--
                        0205  2026
                        0205  2027 UNIT_INIT::                             ; Initialize a DEUNA/DELUA unit
                3F   BB 0205  2028          PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save all regs
      19 64 A5  05   E0 0207  2029          BBS     #UCB$V_POWER,UCB$W_STS(R5),20$ ; Br if powerfail
                        020C  2030
                        020C  2031 ;
                        020C  2032 ; Initialize UCB queue listheads, and the pointer (within the NI device
                        020C  2033 ; dependent UCB extension, UCB$L_NI_MLTPTR) to the multicast address table for
                        020C  2034 ; this protocol type.
                        020C  2035 ;
                        020C  2036
          00E9 C5   9E  020C  2037          MOVAB   UCB$G_XE_MULTI(R5),-    ; Initialize the pointer to this
          0094 C5       0210  2038                  UCB$L_NI_MLTPTR(R5)     ; protocol's multicast address table
                        0213  2039
          50    04   D0 0213  2040          MOVL    #UCB$C_XE_QUEUES,R0     ; Get number of queue listheads in UCB
    51    0098 C5   9E  0216  2041          MOVAB   UCB$Q_XE_QUEUES(R5),R1  ; Get address of queue listheads
          81    61   DE 021B  2042 10$:     MOVAL   (R1),(R1)+              ; Set forward link pointer
      81    FC A1   D0 021E  2043          MOVL    -4(R1),(R1)+            ; Set backward link pointer
            F6 50   F5 0222  2044          SOBGTR  R0,10$                  ; Loop if more listheads
                        0225  2045
                        0225  2046 ;
                        0225  2047 ; If the template UCB, UCB0 is being initialized, query the device ID field of
                        0225  2048 ; PCSR1 and set the device type field of the UCB according to the type of NI
                        0225  2049 ; controller. The default NI controller is the DEUNA. The device type field
                        0225  2050 ; may only be checked if the DEUNA/DELUA has passed the self-test on power-up
                        0225  2051 ; and is in the proper state. If the device is not in the proper state then
                        0225  2052 ; mark UCB0 as being OFFLINE.
                        0225  2053 ;
                        0225  2054
          54 A5   B5    0225  2055 20$:     TSTW    UCB$W_UNIT(R5)          ; Is this unit 0?
             37   12    0228  2056          BNEQ    50$                     ; Branch if not and clear template bit
                04   E1 022A  2057          BBC     #UCB$V_ONLINE,-         ; Do not change the device type if UCB0
       47 64 A5         022C  2058                  UCB$W_STS(R5),60$       ; is offline
                        022F  2059
    51    02 A4   B0    022F  2060          MOVW    PCSR1(R4),R1            ; Copy the contents of PCSR1
```

```
                00   EF   0233  2061          EXTZV   #XE_PCSR1_V_STATE,-     ; Extract and zero extend the port state
      50   51   04        0235  2062                  #XE_PCSR1_S_STATE,R1,R0 ; field of PCSR1
                        0238  2063          $DISPATCH       R0,TYPE=B,-     ; Dispatch based upon the port state
                        0238  2064          < -
                        0238  2065                  <XE_STAT_V_RESET   30$>,-; RESET
                        0238  2066                  <XE_STAT_V_PRIMLD  35$>,-; PRIMARY LOAD
                        0238  2067                  <XE_STAT_V_READY   35$>,-; READY
                        0238  2068                  <XE_STAT_V_RUN     35$>,-; RUN
                        0238  2069                  <XE_STAT_V_UHALT   30$>,-; UNIBUS HALTED
                        0238  2070                  <XE_STAT_V_NIHALT  30$>,-; NI HALTED
                        0238  2071                  <XE_STAT_V_NIUHALT 30$>,-; NI AND UNIBUS HALTED
                        0238  2072                  <XE_STAT_V_PHALT   30$>,-; PORT HALTED (DELUA ONLY)
                        0238  2073          >
                        024E  2074
                        024E  2075  30$:
                04   EF   024E  2076  35$:  EXTZV   #XE_PCSR1_V_DEVID,-     ; Extract and zero extend the device ID
      50   51   03        0250  2077                  #XE_PCSR1_S_DEVID,R1,R0 ; field of PCSR1
                        0253  2078          $DISPATCH       R0,TYPE=B,-     ; Dispatch based upon the device ID
                        0253  2079          < -
                        0253  2080                  <XE_DEVID_V_DEUNA 60$>,-; DEUNA - no change to the type field
                        0253  2081                  <XE_DEVID_V_DELUA 40$>,-; DELUA - change the type field
                        0253  2082          >
            19   90   025B  2083  40$:    MOVB    #DT$_DELUA,-            ; Change the type field to DT$_DELUA
         41 A5        025D  2084                  UCB$B_DEVTYPE(R5)
            15   11   025F  2085          BRB     60$                     ; Branch so template bit is left on
                        0261  2086  ;
                        0261  2087  ;
                        0261  2088  ; We must find the address of unit 0, so we can check if the DEUNA/DELUA is
                        0261  2089  ; ONLINE. If the DEUNA/DELUA is OFFLINE, then we mark each UCB as being OFFLINE.
                        0261  2090  ;
                        0261  2091
          0082 C5   B4  0261  2092  50$:    CLRW    UCB$W_ERRCNT(R5)        ; Only unit #0 may list errors
      50    28 A5   D0  0265  2093          MOVL    UCB$L_DDB(R5),R0        ; Get address of DEUNA/DELUA DDB
      50    04 A0   D0  0269  2094          MOVL    DDB$L_UCB(R0),R0        ; Get address of UNIT 0 UCB
                04   E0  026D  2095          BBS     #UCB$V_ONLINE,-         ; Br if DEUNA/DELUA is ONLINE
         04 64 A0        026F  2096                  UCB$W_STS(R0),60$       ;
                10   AA  0272  2097          BICW    #UCB$M_ONLINE,-         ; Else, mark new unit as OFFLINE
               64 A5        0274  2098                  UCB$W_STS(R5)           ;
                        0276  2099          ;
                        0276  2100          ;  Now check for powerfail! If power had failed, we will ASSUME
                        0276  2101          ;  the DEUNA/DELUA must now be marked ONLINE. This will not
                        0276  2102          ;  work very well if originally the DEUNA/DELUA was offline
                        0276  2103          ;  because it was configured at the wrong CSR (for example).
                        0276  2104          ;
   04 64 A5    05   E1  0276  2105  60$:    BBC     #UCB$V_POWER,UCB$W_STS(R5),65$ ; Br if NOT powerfail recovery
                10   A8  027B  2106          BISW    #UCB$M_ONLINE,-         ; Else, mark UNIT as ONLINE
               64 A5        027D  2107                  UCB$W_STS(R5)
         50    1E   9A  027F  2108  65$:    MOVZBL  #3+MAX_C_MLT,R0         ; Get size of multicast list in words
   51    00E9 C5   9E  0282  2109          MOVAB   UCB$G_XE_MULTI(R5),R1   ; Get address of multicast list
                81   B4  0287  2110  70$:    CLRW    (R1)+                   ; Init multicast table
               FB 50   F5  0289  2111          SOBGTR  R0,70$                  ; Loop if more
                        028C  2112
   54    0180 C5   D0  028C  2113          MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
                06   12  0291  2114          BNEQ    80$                     ; Br if present
              20D8   30  0293  2115          BSBW    ALLOC_CDB               ; Else, try to allocate a CDB
             15 50   E9  0296  2116          BLBC    R0,90$                  ; Br if no error
                        0299  2117
```

```
                        0299  2118 ;
                        0299  2119 ; If a power failure has occured, set the appropriate bit within the CDB
                        0299  2120 ; and schedule a fork process to handle the fatal error.
                        0299  2121 ;
                        0299  2122
   10 64 A5    05   E1  0299  2123 80$:    BBC     #UCB$V_POWER,UCB$W_STS(R5),90$ ; Br if not powerfail
 025A C4    80 8F   88  029E  2124          BISB2   #CDB_STS_M_POWER,CDB_B_STS(R4) ; Set the CDB power failure bit
53   00000100 8F   D0  02A4  2125          MOVL    #XE_PCSR0_M_USCI,R3     ; Indicate a fatal error
            1656   30  02AB  2126          BSBW    SCHED_FORK              ; Schedule a fork process
                   02AE  2127
          3F   BA  02AE  2128 90$:    POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore regs
      64 A5   20   AA  02B0  2129          BICW    #UCB$M_POWER,UCB$W_STS(R5) ; Clear the powerfail flag
               05   02B4  2130          RSB                             ; Done
```

```
                        02B5  2132          .SBTTL  FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
                        02B5  2133 ;++
                        02B5  2134 ; FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
                        02B5  2135 ;
                        02B5  2136 ; Functional description:
                        02B5  2137 ;
                        02B5  2138 ; This routine initializes the FFI interface. Currently the UCB must
                        02B5  2139 ; have been initialized prior to calling this routine, in the future
                        02B5  2140 ; this routine may have to initialize the UCB and DEUNA/DELUA. Therefore,
                        02B5  2141 ; there may be a fork involved in the call to this routine.
                        02B5  2142 ;
                        02B5  2143 ; Inputs:
                        02B5  2144 ;
                        02B5  2145 ;       R3 = Address of quadword descriptor for parameter buffer
                        02B5  2146 ;       R4 = FFI block address
                        02B5  2147 ;
                        02B5  2148 ;       IPL = SYNCH
                        02B5  2149 ;
                        02B5  2150 ; Outputs:
                        02B5  2151 ;
                        02B5  2152 ;       R0 = Status of request
                        02B5  2153 ;       All other registers are preserved.
                        02B5  2154 ;
                        02B5  2155 ;--
                        02B5  2156          ASSUME  IPL$_SYNCH EQ IPL$_XE_FIPL
                        02B5  2157 FFI_INIT::
              3E    BB  02B5  2158          PUSHR   #^M<R1,R2,R3,R4,R5>     ; Save registers
              50    D4  02B7  2159          CLRL    R0                      ; Assume failure
        55    34    A4  02B9  2160          MOVL    FFI$L_DL_UCB(R4),R5     ; Get UCB address
              04    E1  02BD  2161          BBC     #UCB$V_XE_RUN,-         ; Br if device not ready
                        02BF  2162                  UCB$W_DEVSTS(R5),90$
           0389'CF    9E  02C2  2163          MOVAB   W^XMT_FFI_START,-       ; Return address of XMIT routine
              10    A4  02C6  2164                  FFI$L_XMIT(R4)
      0178    C5    54  D0  02C8  2165          MOVL    R4,UCB$L_XE_FFI(R5)     ; Save FFI address
              50    01  9A  02CD  2166          MOVZBL  #1,R0                   ; Return success
                    3E  BA  02D0  2167 90$:     POPR    #^M<R1,R2,R3,R4,R5>     ; Restore registers
                        05  02D2  2168          RSB                             ; Return to caller
                        02D3  2169
```

```
                     02D3  2171          .SBTTL  XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
                     02D3  2172 ;++
                     02D3  2173 ; XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
                     02D3  2174 ;
                     02D3  2175 ; Functional description:
                     02D3  2176 ;
                     02D3  2177 ; This routine sets up the internal function code for transmit and
                     02D3  2178 ; transfers control to the exec buffered I/O write FDT routine.
                     02D3  2179 ;
                     02D3  2180 ; The QIO parameters for WRITES are:
                     02D3  2181 ;
                     02D3  2182 ;       P1 = Address of the data buffer
                     02D3  2183 ;       P2 = Size of the data buffer
                     02D3  2184 ;       P5 = Address of buffer containing the destination address
                     02D3  2185 ;
                     02D3  2186 ; Inputs:
                     02D3  2187 ;
                     02D3  2188 ;       R3 = IRP address
                     02D3  2189 ;       R4 = PCB address
                     02D3  2190 ;       R5 = UCB address
                     02D3  2191 ;       R6 = CCB address
                     02D3  2192 ;       R7 = FUNCTION CODE
                     02D3  2193 ;
                     02D3  2194 ;       IPL = ASTDEL
                     02D3  2195 ;
                     02D3  2196 ; Outputs:
                     02D3  2197 ;
                     02D3  2198 ;       R0-R2,R8,R9 are destroyed.
                     02D3  2199 ;
                     02D3  2200 ;--
                     02D3  2201 ABORTIO_BR:                             ; Long branch to ABORTIO
     00AD     31     02D3  2202          BRW     ABORTIO                ; Abort the I/O request
                     02D6  2203
                     02D6  2204 XMT_FDT::                               ; Transmit FDT routine
          40 A3   7C 02D6  2205          CLRQ    IRP$Q_STATION(R3)      ; Zero the destination address
        00CA C5   B0 02D9  2206          MOVW    UCB$W_XE_PROTYP(R5),-  ; Assume we are a non-promiscuous user
           3A A3      02DD  2207                  IRP$W_XE_PROTYP(R3)    ;
       51  10 AC   D0 02DF  2208          MOVL    P5(AP),R1              ; Get address of destination address
           05   12   02E3  2209          BNEQ    1$                     ; Br if given
                     02E5  2210 ;
                     02E5  2211 ; If the user is in shared mode, then he does not have to supply a destination
                     02E5  2212 ; address with each transmit operation. The destination address will be gotten
                     02E5  2213 ; from the SHR data structure.
                     02E5  2214 ;
           03   E0   02E5  2215          BBS     #UCB$V_XE_SHARE,-      ; Br if shared user
     12 68 A5         02E7  2216                   UCB$W_DEVSTS(R5),2$   ;
        50   0C   9A 02EA  2217 1$:      MOVZBL  S^#SS$_ACCVIO,R0       ; Assume access violation
                     02ED  2218          IFNORD  #6,(R1),ABORTIO_BR     ; Check access to buffer
     40 A3   61   D0 02F3  2219          MOVL    (R1),IRP$Q_STATION(R3) ; Save destination address
44 A3   04 A1   B0 02F7  2220          MOVW    4(R1),IRP$Q_STATION+4(R3) ;     ...
                     02FC  2221
                     02FC  2222          ASSUME  NMA$C_STATE_ON EQ 0
                     02FC  2223          ASSUME  NMA$C_STATE_OFF EQ 1
                     02FC  2224
    0D 00DB C5   E8 02FC  2225 2$:      BLBS    UCB$B_XE_PRM(R5),3$    ; Br if user is not promiscuous
        51   06   C0 0301  2226          ADDL    #6,R1                  ; Point to protocol type
                     0304  2227          IFNORD  #2,(R1),ABORTIO        ; Check access to buffer
```

```
                3A A3   61    B0  030A  2228          MOVW    (R1),IRP$W_XE_PROTYP(R3)        ; Get protocol type from user P5 buf
                   50   14    9A  030E  2229 3$:      MOVZBL  S^#SS$_BADPARAM,R0              ; Assume bad parameters
                   58   6C    D0  0311  2230          MOVL    P1(AP),R8                       ; Get starting address of user buffer
                59    04 AC    3C  0314  2231          MOVZWL  P2(AP),R9                       ; Get length of user buffer
                           69    13  0318  2232          BEQL    ABORTIO                         ; Br if zero length buffer
                   50   58    7D  031A  2233          MOVQ    R8,R0                           ; Retrieve buffer parameters
                00000000'GF   16    031D  2234          JSB     G^EXE$WRITECHK                  ; Check accessibility of user buffer
                                    0323  2235                                                  ; (No return on NO ACCESS)
                                    0323  2236                                                  ; Returns IRP$W_BCNT
             51 0000004C 8F     C0  0323  2237          ADDL2   #CXB$C_HEADER+XE_C_CRC,R1       ; Calculate length of buffer needed
                           38    BB  032A  2238          PUSHR   #^M<R3,R4,R5>                   ; Save registers
                00000000'GF   16    032C  2239          JSB     G^EXE$BUFFRQUOTA                ; Check if process has sufficient quota
                        4C 50     E9  0332  2240          BLBC    R0,10$                          ; Br if quota check failure
                00000000'GF   16    0335  2241          JSB     G^EXE$ALLOCBUF                  ; Allocate CXB buffer for output
                        43 50     E9  033B  2242          BLBC    R0,10$                          ; If LBC allocation failure
                   53   6E    D0  033E  2243          MOVL    (SP),R3                         ; Retrieve address of IRP
             50    0080 C4    D0  0341  2244          MOVL    PCB$L_JIB(R4),R0                ; Get JIB address
               20 A0   51    C2  0346  2245          SUBL    R1,JIB$L_BYTCNT(R0)             ; Adjust buffered I/O quota
               30 A3   51    B0  034A  2246          MOVW    R1,IRP$W_BOFF(R3)               ; Set number of bytes charged to quota
               2C A3   52    D0  034E  2247          MOVL    R2,IRP$L_SVAPTE(R3)             ; Save CXB address in IRP
                       52    DD  0352  2248          PUSHL   R2                              ; Save pointer to CXB
                                 0354  2249
                                 0354  2250          ASSUME  CXB$L_FL EQ 0
                                 0354  2251          ASSUME  CXB$L_BL EQ CXB$L_FL+4
                       82    7C  0354  2252          CLRQ    (R2)+                           ; Clear link cells
                                 0356  2253
                                 0356  2254          ASSUME  CXB$W_SIZE EQ CXB$L_BL+4
                    82   51    B0  0356  2255          MOVW    R1,(R2)+                        ; Set size of structure
                                 0359  2256
                                 0359  2257          ASSUME  CXB$B_TYPE EQ CXB$W_SIZE+2
                                 0359  2258          ASSUME  CXB$B_CODE EQ CXB$B_TYPE+1
                    82   1B    9B  0359  2259          MOVZBW  #DYN$C_CXB,(R2)+                ; Set structure type
                                 035C  2260
                    52   6E    D0  035C  2261          MOVL    (SP),R2                         ; Get back CXB address
                                 035F  2262
                                 035F  2263          ASSUME  CXB$C_HEADER EQ CXB$T_T_DATA+XE_C_HEADER
                 18 A2   3A    B0  035F  2264          MOVW    #CXB$T_T_DATA,CXB$W_BOFF(R2) ; Setup offset to start of data
                 52   48 A2    9E  0363  2265          MOVAB   CXB$C_HEADER(R2),R2             ; Get address of data portion of buffer
                                 0367  2266
             62    68   59    28  0367  2267          MOVC3   R9,(R8),(R2)                    ; Move data to system buffer
                        3C    BA  036B  2268          POPR    #^M<R2,R3,R4,R5>                ; Restore registers
                                 036D  2269          SETIPL  UCB$B_FIPL(R5)                  ; Sync access to UCB
                       53    DD  0371  2270          PUSHL   R3                              ; Save R3
                       4C    10  0373  2271          BSBB    XMT_START                       ; Do common processing
                       53 8ED0   0375  2272          POPL    R3                              ; Restore R3
                     08 50     E9  0378  2273          BLBC    R0,ABORTIO                      ; Br if error in processing request
                00000000'GF   17    037B  2274          JMP     G^EXE$QIORETURN                 ; Exit QIO service to await completion
                                 0381  2275
                       38    BA  0381  2276 10$:     POPR    #^M<R3,R4,R5>                   ; Restore registers
                00000000'GF   17    0383  2277 ABORTIO:JMP     G^EXE$ABORTIO                   ; Abort the I/O request
```

```
                     0389  2279          .SBTTL  XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
                     0389  2280 ;++
                     0389  2281 ; XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
                     0389  2282 ;
                     0389  2283 ; Functional description:
                     0389  2284 ;
                     0389  2285 ; This routine is called to start a transmit operation. If the DEUNA/DELUA is
                     0389  2286 ; running then the request is given to the xmit wait queue for the DEUNA/DELUA.
                     0389  2287 ; If there is a free entry in the transmit ring and there are sufficient map
                     0389  2288 ; registers to map the buffer then the request is given to the DEUNA/DELUA
                     0389  2289 ; immediately, else the request is left on the xmit wait queue until another
                     0389  2290 ; request completes.
                     0389  2291 ;
                     0389  2292 ;
                     0389  2293 ;  ** The driver should never do direct I/O on XMIT requests, because  **
                     0389  2294 ; *** the DEUNA/DELUA is very slow if the buffer address begins on an odd byte ***
                     0389  2295 ;  ** boundary. Also, the FAST interface cannot operate on DIRECT I/O. **
                     0389  2296 ;
                     0389  2297 ; Inputs:
                     0389  2298 ;
                     0389  2299 ;       R3 = CXB address
                     0389  2300 ;       R4 = FFI address
                     0389  2301 ;
                     0389  2302 ;       IPL = SYNCH (same as FIPL)
                     0389  2303 ;
                     0389  2304 ; Outputs:
                     0389  2305 ;
                     0389  2306 ;       R0,R3 are destroyed.
                     0389  2307 ;       All other registers are preserved.
                     0389  2308 ;
                     0389  2309 ;       If the request cannot be queued, then the FFI$L_XMIT_DONE entry is
                     0389  2310 ;       called immediately with the following:
                     0389  2311 ;
                     0389  2312 ;               R0 = Status of transmit request
                     0389  2313 ;               R3 = CXB address
                     0389  2314 ;               R4 = FFI address
                     0389  2315 ;
                     0389  2316 ;--
                     0389  2317          ASSUME  IPL$_SYNCH EQ IPL$_XE_FIPL
                     0389  2318 XMT_FFI_START::                          ; Start FAST interface transmit request
          36   BB    0389  2319          PUSHR   #^M<R1,R2,R4,R5>         ; Save registers
    55    34   A4    D0    038B  2320          MOVL    FFI$L_DL_UCB(R4),R5      ; Get UCB address
          24   A3    D4    038F  2321          CLRL    CXB$L_T_IRP(R3)          ; No IRP with CXB
    18 A3     0E    A2    0392  2322          SUBW    #XE_C_HEADER,CXB$W_BOFF(R3) ; Back up offset for Ethernet header
                           0396  2323
                           0396  2324          ASSUME  NMA$C_STATE_ON EQ 0
                           0396  2325          ASSUME  NMA$C_STATE_OFF EQ 1
    04 00D9   C5    E8    0396  2326          BLBS    UCB$B_XE_PAD(R5),20$     ; Br if padding is disabled
    18 A3     02    A2    039B  2327          SUBW    #XE_C_CNTSIZ,CXB$W_BOFF(R3) ; Else, skip length field of buffer
    51   18   A3    3C    039F  2328 20$:     MOVZWL  CXB$W_BOFF(R3),R1        ; Get offset to start of data
52  53    51    C1    03A3  2329          ADDL3   R1,R3,R2                 ; Set R2 to start of header
                           03A7  2330
                           03A7  2331          ASSUME  XBUF_W_UNUSED EQ XBUF_G_DEST+6
          28   A3    7D    03A7  2332          MOVQ    CXB$Q_STATION(R3),-      ; Store destination address
                62          03AA  2333                   XBUF_G_DEST(R2)
       00CA   C5    B0    03AB  2334          MOVW    UCB$W_XE_PROTYP(R5),-    ; Set PROTOCOL TYPE
         0C   A2          03AF  2335                   XBUF_W_TYPE(R2)
```

```
                49  10  03B1  2336          BSBB    XMT_INITIATE                ; Try to start transmit
             08 50  E8  03B3  2337          BLBS    R0,50$                      ; Br if success
     54   0178 C5  D0  03B6  2338          MOVL    UCB$L_XE_FFI(R5),R4         ; Else, get back FFI address
             14 B4  16  03BB  2339          JSB     @FFI$L_XMIT_DONE(R4)        ; Complete request in error
                36  BA  03BE  2340 50$:     POPR    #^M<R1,R2,R4,R5>            ; Restore registers
                    05  03C0  2341          RSB                                 ; Return to caller
```

```
                                        .SBTTL  XMT_START - START TRANSMIT OPERATION
                                ;++
                                ; XMT_START - START TRANSMIT OPERATION
                                ;
                                ; Functional description:
                                ;
                                ; This routine is called to start a transmit operation. If the DEUNA/DELUA is
                                ; running then the request is given to the xmit wait queue for the DEUNA/DELUA.
                                ; If there is a free entry in the transmit ring and there are sufficient map
                                ; registers to map the buffer then the request is given to the DEUNA/DELUA
                                ; immediately, else the request is left on the xmit wait queue until another
                                ; request completes.
                                ;
                                ;
                                ;  ** The driver should never do direct I/O on XMIT requests, because  **
                                ; *** the DEUNA/DELUA is very slow if the buffer address begins on an odd byte ***
                                ;  ** boundary. Also, the FAST interface cannot operate on DIRECT I/O. **
                                ;
                                ; Inputs:
                                ;
                                ;       R2 = CXB address
                                ;       R3 = IRP address
                                ;       R5 = UCB address
                                ;
                                ;       IPL = FIPL
                                ;
                                ; Outputs:
                                ;
                                ;       R0 = Status of transmit request
                                ;       R1,R2,R3,R4 are destroyed.
                                ;
                                ;--
                                        .ENABL  LSB
                                XMT_START::                                     ; Start transmit operation
            04      E0                  BBS     #UCB$V_XE_RUN,-                 ; Br if unit is in RUN mode
        10 68       A5                          UCB$W_DEVSTS(R5),30$
                                ;
 50       20D4       8F      3C      MOVZWL  #SS$_DEVINACT,R0                    ; Assume unit not started yet
                    0E      E1      BBC     #UCB$V_XE_INTERLOCK,-               ; Br if unit is not re-starting
        05 68       A5                  UCB$W_DEVSTS(R5),20$                    ;  on it's own.
 50       02D4       8F      3C      MOVZWL  #SS$_OPINCOMPL,R0                   ; Else, return different error code
                            05      RSB                                         ; Okay to leave now
                                ;
   24 A2            53      D0      MOVL    R3,CXB$L_T_IRP(R2)                  ; Save IRP address
1A A2               32      A3      MOVW    IRP$W_BCNT(R3),CXB$W_BCNT(R2) ; Set BCNT in CXB
        53          52      7D      MOVQ    R2,R3                               ; Copy CXB and IRP addresses
                                ;
                                        ASSUME  NMA$C_STATE_ON EQ 0
                                        ASSUME  NMA$C_STATE_OFF EQ 1
   04 00D9          C5      E8      BLBS    UCB$B_XE_PAD(R5),35$        ; Br if padding is disabled
   18 A3            02      A2      SUBW    #XE_C_CNTSIZ,CXB$W_BOFF(R3) ; Else, skip length field of buffer
   51       18 A3           3C      MOVZWL  CXB$W_BOFF(R3),R1           ; Get offset to start of data
 52    53          51      C1      ADDL3   R1,R3,R2                    ; Set R2 to start of header
          40 A4     7D      MOVQ    IRP$Q_STATION(R4),-         ; Store destination address
                62          03F6                                XBUF_G_DEST(R2)
          3A A4     B0      03F7    MOVW    IRP$W_XE_PROTYP(R4),-       ; Store PROTOCOL TYPE
          0C A2             03FA                                XBUF_W_TYPE(R2)
```

```
                        03FC  2400
                        03FC  2401 XMT_INITIATE::                          ; FAST Interface entry point (FFI)
                        03FC  2402          ;
                        03FC  2403          ; Inputs:
                        03FC  2404          ;       R5 = UCB address
                        03FC  2405          ;       R3 = CXB address
                        03FC  2406          ;       R2 = Start address for Ethernet header
                        03FC  2407          ;
20 A3    07    90       03FC  2408          MOVB    #XE_FC_V_XMIT,CXB$B_XE_FUNC(R3) ; Set function request in CXB
08 A2    55    D0       0400  2409          MOVL    R5,XBUF_L_UCB(R2)          ; Save UCB address
         50    D4       0404  2410          CLRL    R0                         ; Assume no CRC
                        0406  2411
                        0406  2412          ASSUME  NMA$C_STATE_ON EQ 0
                        0406  2413          ASSUME  NMA$C_STATE_OFF EQ 1
03 00DD  C5    E9       0406  2414          BLBC    UCB$B_XE_CRC(R5),40$       ; Br if CRC generation is ON
   50    04    B0       040B  2415          MOVW    #XE_C_CRC,R0               ; Else, allow user to xmit a CRC
                        040E  2416
                        040E  2417 40$:     ASSUME  NMA$C_STATE_ON EQ 0
                        040E  2418          ASSUME  NMA$C_STATE_OFF EQ 1
09 00D9  C5    E8       040E  2419          BLBS    UCB$B_XE_PAD(R5),60$       ; Br if padding is disabled
                        0413  2420
                        0413  2421 ;
                        0413  2422 ;  PADDING IS ENABLED:
                        0413  2423 ;       Adjust byte count to include size field and store count field.
                        0413  2424 ;
     1A  A3    B0       0413  2425          MOVW    CXB$W_BCNT(R3),-           ; Else, store size of data-only
   0E  A2             0416  2426                  XBUF_W_SIZE(R2)            ;  portion of buffer in message
1A A3    02    A0       0418  2427          ADDW    #XE_C_CNTSIZ,CXB$W_BCNT(R3) ; And account for count field
                        041C  2428
                        041C  2429 ;
                        041C  2430 ; Allow buffer size up to Ethernet max buffer size for transmit operations.
                        041C  2431 ;
51   50    05DC  8F    A1  041C  2432 60$:     ADDW3   #MAX_PKT_SIZE,R0,R1        ; Calculate size of largest packet
     51     1A  A3    B1  0422  2433          CMPW    CXB$W_BCNT(R3),R1          ; Is buffer size bigger than
                        0426  2434                                             ;  largest Ethernet buffer allowed?
               06    1B  0426  2435          BLEQU   90$                        ; Br if no
     50    034C  8F    3C  0428  2436 80$:     MOVZWL  #SS$_IVBUFLEN,R0           ; Assume bad buffer length
               05    05  042D  2437          RSB                                ; ELSE, leave now
                        042E  2438
     54    0180  C5    D0  042E  2439 90$:     MOVL    UCB$L_XE_CDB(R5),R4        ; Get CDB address
               1D    13  0433  2440          BEQL    100$                       ; Return error if no CDB
               01    E1  0435  2441          BBC     #CDB_STS_V_RUN,-           ; Br if DEUNA/DELUA not running
         17  025A  C4        0437  2442                  CDB_B_STS(R4),100$         ;
               0E    A0  043B  2443          ADDW    #XE_C_HEADER,-             ; Adjust byte count
                  1A  A3        043D  2444                  CXB$W_BCNT(R3)             ;  for header info
                        043F  2445 ;
                        043F  2446 ; If running in the SHARED-LIMITED mode, then we must use the destination
                        043F  2447 ; address from the SHR_ data structure. Unless the given destination address
                        043F  2448 ; is a multicast address. For the SHARED-DEFAULT user, we must make sure that
                        043F  2449 ; destination address given is unique!
                        043F  2450 ;
               03    E1  043F  2451          BBC     #UCB$V_XE_SHARE,-          ; Br if NOT a shared user
            4A  68  A5        0441  2452                  UCB$W_DEVSTS(R5),NO_SHR ;
                        0444  2453 ;
                        0444  2454 ; Try to find a match on PID/CHAN. Returns pointer in R1
                        0444  2455 ;
               53    DD  0444  2456          PUSHL   R3                         ; Save CXB address
```

```
                53   24 A3    D0  0446  2457          MOVL    CXB$L_T_IRP(R3),R3      ; Get IRP address
                     01E0     30  044A  2458          BSBW    MATCH_SHR               ; Try to find the SHR data structure
                       06     13  044D  2459          BEQL    110$                    ; Br if match
                       53  8ED0  044F  2460          POPL    R3                      ; Else, restore CXB address
                     FF71     31  0452  2461 100$:    BRW     10$                     ; Return error
                              0455  2462
                       53  8ED0  0455  2463 110$:    POPL    R3                      ; Restore CXB address
                00C4   C5    D1  0458  2464          CMPL    UCB$L_XE_DEFUSR(R5),R1  ; Is this the default user?
                       0E    13  045D  2465          BEQL    SHR_DEF                 ; Br if yes
                             045F  2466 ;
                             045F  2467 ; This is a SHARED-LIMITED user.
                             045F  2468 ;
                     2C 62   E8  045F  2469          BLBS    XBUF_G_DEST(R2),NO_SHR  ; Br if multicast address
                     12 A1   D0  0462  2470          MOVL    SHR_G_DEST(R1),-        ; Else, get destination from SHR struct.
                        62       0465  2471                   XBUF_G_DEST(R2)         ;       ...
                     16 A1   B0  0466  2472          MOVW    SHR_G_DEST+4(R1),-      ;       ...
                     04 A2       0469  2473                   XBUF_G_DEST+4(R2)       ;       ...
                        21   11  046B  2474          BRB     NO_SHR                  ; Continue in common code
                             046D  2475          .DSABL  LSB
                             046D  2476 ;
                             046D  2477 ; This is a SHARED-DEFAULT user.
                             046D  2478 ;
                0098   C5    9E  046D  2479 SHR_DEF:MOVAB   UCB$Q_XE_SHARE(R5),R0   ; Get address of SHR listhead
                     51   50   D0  0472  2480          MOVL    R0,R1                   ; Save address of start of listhead
                     51   61   D0  0475  2481 10$:     MOVL    SHR_L_QFL(R1),R1        ; Get address of next in list
                     51   50   D1  0478  2482          CMPL    R0,R1                   ; Back at start of list?
                        11   13  047B  2483          BEQL    NO_SHR                  ; Br if yes, destination is unique
                        62   D1  047D  2484          CMPL    XBUF_G_DEST(R2),-       ; Address match?
                     12 A1       047F  2485                   SHR_G_DEST(R1)          ;       ...
                        F2   12  0481  2486          BNEQ    10$                     ; Br if no - check next in list
                     04 A2   B1  0483  2487          CMPW    XBUF_G_DEST+4(R2),-     ; Hi order of address match?
                     16 A1       0486  2488                   SHR_G_DEST+4(R1)        ;       ...
                        EB   12  0488  2489          BNEQ    10$                     ; Br if not - check next in list
                     50   14    3C  048A  2490          MOVZWL  #SS$_BADPARAM,R0        ; Else, bad parameter code
                          05    RSB  048D  2491          RSB                             ; Return to caller
                             048E  2492
                             048E  2493 NO_SHR:
                             048E  2494 ;
                             048E  2495 ; If we are running in point-to-point mode, then queue xmit on wait queue
                             048E  2496 ; if we are waiting for run!
                             048E  2497 ;
                        00   91  048E  2498          CMPB    #NMA$C_LINPR_POI,-      ;% Are we in PT-TO-PT mode?
                00D8   C5       0490  2499                   UCB$B_XE_PRO(R5)        ;%
                        0E   12  0493  2500          BNEQ    40$                     ;% Br if not
                        06   E1  0495  2501          BBC     #UCB$V_XE_STACK,-       ;% Br if not in stack wait state
                     09 68   A5       0497  2502                   UCB$W_DEVSTS(R5),40$    ;%
                00B4   D5    63  049A  2503          INSQUE  (R3),@UCB$Q_XE_XMTREQ+4(R5);% Else, insert request on wait queue
                     50   01   9A  049F  2504          MOVZBL  #1,R0                   ;% Return success
                          05    RSB  04A2  2505          RSB                             ;% Return to caller
                             04A3  2506 ;
                             04A3  2507 ; Insert request on CDB transmit request queue and check if transmit
                             04A3  2508 ; can proceed.
                             04A3  2509 ;
                0080   D4    63  04A3  2510 40$:     INSQUE  (R3),@CDB_Q_XMTREQ+4(R4) ; Insert at end of xmit queue
                             04A8  2511
                             04A8  2512 XMT_ALT_START::                              ; Alternate start for xmit
                             04A8  2513          ;
```

```
                                04A8  2514          ; Inputs:
                                04A8  2515          ;       R5 = UCB
                                04A8  2516          ;       R4 = CDB
                                04A8  2517          ;
                                04A8  2518          PUSHQ   R6                              ; Save R6,R7
                     56   24 A5   D0  04AB  2519          MOVL    UCB$L_CRB(R5),R6        ; Get CRB address
                                04AF  2520          ASSUME  MAX_C_XMT LE 8
          57   1B A4   04   00   EB  04AF  2521          FFC     #0,#MAX_C_XMT,CDB_B_XMTMAP(R4),R7 ; Find a free transmit slot
                           43   13  04B5  2522          BEQL    20$                     ; Br if none free
                                04B7  2523 ;
                                04B7  2524 ; Move CXB info into UCB
                                04B7  2525 ;
                     53   7C B4   0F  04B7  2526          REMQUE  @CDB_Q_XMTREQ(R4),R3    ; Get oldest xmit request
                                3D   1D  04BB  2527          BVS     20$                     ; Br if none
                   7E A5   1A A3   B0  04BD  2528          MOVW    CXB$W_BCNT(R3),UCB$W_BCNT(R5) ; Set byte count
                     52   18 A3   3C  04C2  2529          MOVZWL  CXB$W_BOFF(R3),R2       ; Get offset to start of data
                          52   53   C0  04C6  2530          ADDL    R3,R2                   ; Compute buffer virtual address
                        FE00 8F   AB  04C9  2531          BICW3   #^C<VA$M_BYTE>,-        ; Get buffer offset
                     7C A5   52       04CD  2532                  R2,UCB$W_BOFF(R5)       ;
                                04D0  2533 ;
                                04D0  2534 ; Convert virtual address to physical PTE address
                                04D0  2535 ;
          52   52   15   09   EF  04D0  2536          EXTZV   #VA$V_VPN,#VA$S_VPN,R2,R2 ; Get virtual page number
           50   00000000'GF   D0  04D5  2537          MOVL    G^MMG$GL_SPTBASE,R0     ; Get the base address of the SPT
                78 A5   6042   DE  04DC  2538          MOVAL   (R0)[R2],UCB$L_SVAPTE(R5) ; Set address of the SPT entry
                                04E1  2539 ;
                                04E1  2540 ; The following instruction also sets the data path number to the Direct
                                04E1  2541 ; Data Path.
                                04E1  2542 ;
                                04E1  2543          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                04E1  2544          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                     3C A4   D0  04E1  2545          MOVL    CDB_L_XMTMAP(R4),-      ; Assume we use preallocated map
                     34 A6       04E4  2546                  CRB$L_INTD+VEC$W_MAPREG(R6) ; register.
                        57   D5  04E6  2547          TSTL    R7                      ; Is mapping slot the preallocated one?
                        17   13  04E8  2548          BEQL    50$                     ; Br if yes - all set
                                04EA  2549                                          ; Else, allocate the map registers
                                04EA  2550 ;
                                04EA  2551 ; Allocate UNIBUS map registers
                                04EA  2552 ;
                     37 A6   94  04EA  2553          CLRB    CRB$L_INTD+VEC$B_DATAPATH(R6) ; Reset data path usage
          00000000'GF   16  04ED  2554          JSB     G^IOC$ALOUBAMAP         ; Allocate UNIBUS map registers
                     0B 50   E8  04F3  2555          BLBS    R0,50$                  ; Br if one available
                7C A4   63   0E  04F6  2556          INSQUE  (R3),CDB_Q_XMTREQ(R4)   ; Re-insert CXB on request queue
                                04FA  2557 20$:      POPQ    R6                      ; Restore R6,R7
                   50   01   9A  04FD  2558          MOVZBL  S^#SS$_NORMAL,R0        ; Good return
                                05  0500  2559 30$:      RSB                             ; Return to caller
                                0501  2560 ;
                                0501  2561 ; Try to allocate a buffered data path. If successful use it, else use the
                                0501  2562 ; Direct Data Path.
                                0501  2563 ;
                                0501  2564 ; This driver does not pre-allocate any buffered data paths because it is
                                0501  2565 ; not known how many DEUNAs/DELUAs will be installed on any one CPU. Neither is
                                0501  2566 ; it known how many other devices on that same CPU may also wish to use the
                                0501  2567 ; buffered data paths, therefore no pre-allocation is done.
                                0501  2568 ;
          00000000'GF   16  0501  2569 50$:      JSB     G^IOC$REQDATAPNW        ; Request a buffered data path
                                0507  2570 ;
```

```
                                                0507  2571 ; Save the map information and map the buffer.
                                                0507  2572 ;
                                                0507  2573         ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                                0507  2574         ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                                  34 A6   D0    0507  2575         MOVL    CRB$L_INTD+VEC$W_MAPREG(R6),- ; Save mapping info
                                3C A447         050A  2576                 CDB_L_XMTMAP(R4)[R7]    ; in CDB
                                                050D  2577         SETBIT  R7,CDB_B_XMTMAP(R4)     ; Set mapping slot in use flag
                              22 A3   57   90   0512  2578         MOVB    R7,CXB$B_XE_SLOT(R3)    ; Save mapping slot number used
                                                0516  2579 ;
                                                0516  2580 ; Find next ring entry and insert data
                                                0516  2581 ;
                              52   18 A4   9A   0516  2582         MOVZBL  CDB_B_NEXTXMT(R4),R2    ; Get next ring entry
                                   18 A4   96   051A  2583         INCB    CDB_B_NEXTXMT(R4)       ; Bump ring pointer
                                   FC 8F   8A   051D  2584         BICB    #^C<MAX_C_XMT-1>,-      ; Modulo xmit ring size
                                   18 A4        0520  2585                 CDB_B_NEXTXMT(R4)       ;
                                23 A3   52   90   0522  2586         MOVB    R2,CXB$B_XE_RING(R3)    ; Save ring entry number
                             52    4C A442   D0   0526  2587         MOVL    CDB_L_XRADDR(R4)[R2],R2 ; Retrieve next ring entry address
                                62   1A A3   B0   052B  2588         MOVW    CXB$W_BCNT(R3),XMT_W_SLEN(R2) ; Store message length
                             08 A2   22 A3   B0   052F  2589         MOVW    CXB$W_XE_RID(R3),XMT_W_RID(R2) ; Store unique request ID
                             02 A2   7C A5   B0   0534  2590         MOVW    UCB$W_BOFF(R5),XMT_W_SEGBL(R2) ; Move byte offset - BA0-BA8
                                   34 A6   F0   0539  2591         INSV    CRB$L_INTD+VEC$W_MAPREG(R6),- ; Insert BA9-BA15
                          02 A2    07   09        053C  2592                 #9,#7,XMT_W_SEGBL(R2)   ;&
                    50    34 A6    02   07   EF   0540  2593         EXTZV   #7,#2,CRB$L_INTD+VEC$W_MAPREG(R6),R0 ; Get BA16-BA17
                             04 A2   50   90   0546  2594         MOVB    R0,XMT_B_SEGBH(R2)      ; Insert BA16-BA17
                                56   52   D0   054A  2595         MOVL    R2,R6                   ; Save xmit ring entry
                              00000000'GF   16   054D  2596         JSB     G^IOC$LOADUBAMAPA       ; Load map registers
                             05 A6   83 8F   90   0553  2597         MOVB    #XMT_FLG_M_STP!XMT_FLG_M_ENP!- ; Give buffer to DEUNA/DELUA
                                                0558  2598                 XMT_FLG_M_OWN,XMT_B_FLAGS(R6) ;
                                   00B7 C4   96   0558  2599         INCB    CDB_B_XMTCNT(R4)        ; Tally one more xmit in progress
                                                055C  2600 ;
                                                055C  2601 ; Request and load the port
                                                055C  2602 ;
                                                055C  2603         POPQ    R6                      ; Restore R6,R7
                                                055F  2604         DSBINT  UCB$B_DIPL(R5)          ; Sync access to device
                    0088 D4     63   0E   0566  2605         INSQUE  (R3),@CDB_Q_INPUT+4(R4) ; Insert at end of input queue
                                03   12   056B  2606         BNEQ    60$                     ; Br if not first entry on queue
                              0F4C   30   056D  2607         BSBW    LOAD_PORT               ; Load port
                                                0570  2608 60$:    ENBINT                  ; Restore IPL
                              FF32   31   0573  2609         BRW     XMT_ALT_START           ; Try for more requests
                                                0576  2610
```

```
                                                J 2
0576  2612          .SBTTL  RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
0576  2613 ;++
0576  2614 ; RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
0576  2615 ;
0576  2616 ; Functional description:
0576  2617 ;
0576  2618 ; The specified buffer is checked for accessibility. The buffer address and count
0576  2619 ; are saved in the packet. Then IPL is set to device fork IPL and if a message is
0576  2620 ; available the operation is completed; otherwise, the packet is queued onto
0576  2621 ; the waiting receive list.
0576  2622 ;
0576  2623 ; The QIO parameters for WRITES are:
0576  2624 ;       P1 = Address of the data buffer
0576  2625 ;       P2 = Size of the data buffer
0576  2626 ;       P5 = Optional address of the buffer to receive the source address
0576  2627 ;
0576  2628 ; Inputs:
0576  2629 ;
0576  2630 ;       R3 = IRP address
0576  2631 ;       R4 = PCB address
0576  2632 ;       R5 = UCB address
0576  2633 ;       R6 = CCB address
0576  2634 ;       R7 = Function code
0576  2635 ;       AP = Address of the first operation specific qio parameter
0576  2636 ;
0576  2637 ;       IPL = ASTDEL
0576  2638 ;
0576  2639 ; Outputs:
0576  2640 ;
0576  2641 ;       R0 = Status of the receive qio operation
0576  2642 ;       R3 = IRP address
0576  2643 ;       R5 = UCB ADDRESS
0576  2644 ;
0576  2645 ;       R1,R2 are destroyed.
0576  2646 ;--
0576  2647
0576  2648 RCV_FDT::                                    ; Read operation FDT
0576  2649 ;
0576  2650 ;  Check the request params
0576  2651 ;
      30 A3   B4  0576  2652          CLRW    IRP$W_BOFF(R3)          ; Set no quota to here
   50    0C   9A  0579  2653          MOVZBL  S^#SS$_ACCVIO,R0        ; Assume access violation
57    10 AC   D0  057C  2654          MOVL    P5(AP),R7               ; Get address for source address
         33   13  0580  2655          BEQL    10$                     ; Br if none
                  0582  2656          IFNOWRT #RHDR_C_DATA,(R7),20$   ; Check for write access to buffer
         07   E0  0588  2657          BBS     #IRP$V_DIAGBUF,-        ; Br if diagnostic buffer given
   28 2A A3         058A  2658                  IRP$W_STS(R3),10$       ;       ...
   51    1A   9A  058D  2659          MOVZBL  #RHDR_C_LENGTH,R1       ; Get size of header buffer
         53   DD  0590  2660          PUSHL   R3                      ; Save IRP address
00000000'GF    16  0592  2661          JSB     G^EXE$ALLOCBUF          ; Allocate header buffer
         53 8ED0  0598  2662          POPL    R3                      ; Restore IRP address
   40 50   E9  059B  2663          BLBC    R0,20$                  ; Br if allocation failure
4C A3    52   D0  059E  2664          MOVL    R2,IRP$L_DIAGBUF(R3)    ; Save buffer address
2A A3  0080 8F   A8  05A2  2665          BISW    #IRP$M_DIAGBUF,IRP$W_STS(R3) ; Indicate diag buffer present
                     05A8  2666          ASSUME  RHDR_L_DATA EQ 0
82    0C A2   9E  05A8  2667          MOVAB   RHDR_T_DATA(R2),(R2)+   ; Set address of start of data
                     05AC  2668          ASSUME  RHDR_L_BUFFER EQ RHDR_L_DATA+4
```

```
                82    57    D0  05AC  2669          MOVL    R7,(R2)+                ; Set user buffer address
                                05AF  2670          ASSUME  RHDR_W_SIZE EQ RHDR_L_BUFFER+4
                82    51    B0  05AF  2671          MOVW    R1,(R2)+                ; Save size of allocation
                                05B2  2672          ASSUME  RHDR_B_TYPE EQ RHDR_W_SIZE+2
                82    13    90  05B2  2673          MOVB    #DYN$C_BUFIO,(R2)+      ; Set structure type
                50    14    9A  05B5  2674 10$:     MOVZBL  S^#SS$_BADPARAM,R0      ; Assume illegal size
              51    04 AC   3C  05B8  2675          MOVZWL  P2(AP),R1               ; Get size
                                05BC  2676
                                05BC  2677
                                05BC  2678
                      20    13  05BC  2679          BEQL    20$                     ; Br if zero - illegal
                50    6C    D0  05BE  2680          MOVL    P1(AP),R0               ; Get user buffer address
             3C A3    50    D0  05C1  2681          MOVL    R0,IRP$L_XE_DATBUF(R3)  ; Save user VA for completion
          00000000'GF    16  05C5  2682          JSB     G^EXE$READCHK           ; Check the buffer
                                05CB  2683                                          ; (No return on NO ACCESS)
             2A A3    20    A8  05CB  2684          BISW    #IRP$M_CHAINED,IRP$W_STS(R3) ; Allow data chaining
                                05CF  2685          SETIPL  UCB$B_FIPL(R5)          ; Raise IPL to lock data base
                      12    10  05D3  2686          BSBB    RCV_START               ; Process the request
                   06 50    E9  05D5  2687          BLBC    R0,20$                  ; Br if error
          00000000'GF    17  05D8  2688          JMP     G^EXE$QIORETURN         ; Else, take normal return
                                05DE  2689
                   FDA2    31  05DE  2690 20$:     BRW     ABORTIO                 ; Abort the request
```

```
                     05E1  2692          .SBTTL  RCV_START - START RECEIVE I/O OPERATION
                     05E1  2693 ;++
                     05E1  2694 ; RCV_START - START RECEIVE I/O OPERATION
                     05E1  2695 ;
                     05E1  2696 ; Functional description:
                     05E1  2697 ;
                     05E1  2698 ;  Check for device active.  Receives cannot be queued to the UCB receive
                     05E1  2699 ;  queue unless the UCB has been initialized via routine START.   They
                     05E1  2700 ;  cannot be put in the IRP queue since this could result in non-sequential
                     05E1  2701 ;  receive processing due to the existence of the separate receive queue.
                     05E1  2702 ;
                     05E1  2703 ; Inputs:
                     05E1  2704 ;
                     05E1  2705 ;       R3 = IRP address
                     05E1  2706 ;       R5 = UCB address
                     05E1  2707 ;
                     05E1  2708 ;       IPL = FIPL
                     05E1  2709 ;
                     05E1  2710 ; Outputs:
                     05E1  2711 ;
                     05E1  2712 ;       R0 = Status of receive request
                     05E1  2713 ;       R4 = CDB address
                     05E1  2714 ;
                     05E1  2715 ;       R1,R2 are destroyed.
                     05E1  2716 ;
                     05E1  2717 ;--
                     05E1  2718 INACT_ERROR:
          50  20D4 8F  3C  05E1  2719          MOVZWL  #SS$_DEVINACT,R0        ; Setup return status
                       05  05E6  2720          RSB                             ; Return to caller
                     05E7  2721
                     05E7  2722          .ENABL  LSB
                     05E7  2723 RCV_START::                                  ; Start receive I/O operation
                04   E1  05E7  2724          BBC     #UCB$V_XE_RUN,-         ; Br if UCB is not in RUN mode
              F5 68 A5      05E9  2725                  UCB$W_DEVSTS(R5),INACT_ERROR
                     05EC  2726
          54  0180 C5  D0  05EC  2727          MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
                39   13  05F1  2728          BEQL    40$                     ; Br if none
                01   E1  05F3  2729          BBC     #CDB_STS_V_RUN,-        ; Br if DEUNA/DELUA not running
            E8 025A C4      05F5  2730                  CDB_B_STS(R4),INACT_ERROR
          51  00A0 C5  9E  05F9  2731          MOVAB   UCB$Q_XE_RCVMSG(R5),R1  ; Get address of UCB received messages
                     05FE  2732 ;
                     05FE  2733 ; If running in SHARED mode, then use the listheads in the SHR_
                     05FE  2734 ; data structure.
                     05FE  2735 ;
                03   E1  05FE  2736          BBC     #UCB$V_XE_SHARE,-       ; Br if UCB is NOT SHARED
              09 68 A5      0600  2737                  UCB$W_DEVSTS(R5),5$     ;
                     0603  2738 ;
                     0603  2739 ; Try to find a match on PID/CHAN
                     0603  2740 ;
                0027 30  0603  2741          BSBW    MATCH_SHR               ; Try to find shared user
                  D9 12  0606  2742          BNEQ    INACT_ERROR             ; Br if none - inactive user
            51  18 A1  9E  0608  2743          MOVAB   SHR_Q_RCVMSG(R1),R1     ; Get address of received messages
                     060C  2744 ;
                     060C  2745 ; Check to see if message is available
                     060C  2746 ;
              52  91  0F  060C  2747 5$:       REMQUE  @(R1)+,R2               ; Dequeue a received message
                     060F  2748                                              ; ..bump pointer to end of list pointer
```

```
                 05   1D  060F  2749          BVS     10$                     ; Br if none
                          0611  2750 ;
                          0611  2751 ; Complete receive with available message
                          0611  2752 ;
                185A   30 0611  2753          BSBW    FINISH_RCV_IO           ; Complete the receive
                  13   11 0614  2754          BRB     30$                     ; And exit
                          0616  2755 ;
                          0616  2756 ; Queue IRP for future message arivial unless IO$M_NOW specified
                          0616  2757 ;
          0A 20 A3   06   E1 0616  2758 10$:    BBC     #IO$V_NOW,IRP$W_FUNC(R3),20$ ; Br if not READ NOW
           50   0870 8F   3C 061B  2759          MOVZWL  #SS$_ENDOFFILE,R0       ; Set no message status
                1928   30 0620  2760          BSBW    IO_DONE                 ; Complete the I/O
                  04   11 0623  2761          BRB     30$                     ; And exit
                          0625  2762 ;
                          0625  2763 ; Queue the IRP to UCB receive wait queue
                          0625  2764 ;
                          0625  2765          ASSUME  UCB$Q_XE_RCVREQ EQ UCB$Q_XE_RCVMSG+8
                          0625  2766          ASSUME  SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
             08 B1   63   0E 0625  2767 20$:    INSQUE  (R3),@8(R1)             ; Put packet on waiting list
                50   01   9A 0629  2768 30$:    MOVZBL  S^#SS$_NORMAL,R0        ; Set QIO status return
                          05 062C  2769 40$:    RSB                             ; Return to caller
                          062D  2770          .DSABL  LSB
```

```
                        062D  2772          .SBTTL   SUBROUTINES TO FIND SHR DATA STRUCTURE
                        062D  2773 ;+
                        062D  2774 ; Subroutine to find SHR data structure for user
                        062D  2775 ;
                        062D  2776 ; Inputs:
                        062D  2777 ;       R3 = Address of IRP
                        062D  2778 ;       R5 = UCB address
                        062D  2779 ;
                        062D  2780 ; Outputs:
                        062D  2781 ;       R1 = Address if SHR data structure if match
                        062D  2782 ;       R0 is destroyed.
                        062D  2783 ;       Z-Bit set then match.
                        062D  2784 ;       Z-Bit clear then no match.
                        062D  2785 ;-
                        062D  2786
                        062D  2787 MATCH_SHR:                                   ; Try to find shared user
51   00C4 C5    D0      062D  2788          MOVL    UCB$L_XE_DEFUSR(R5),R1     ; Get address of default user
           04    13      0632  2789          BEQL    10$                        ; Br if no default user
           1C    10      0634  2790          BSBB    CHECK_SHR                  ; Check for match
           19    13      0636  2791          BEQL    40$                        ; Br if match
50   0098 C5    9E      0638  2792 10$:     MOVAB   UCB$Q_XE_SHARE(R5),R0      ; Save address of listhead
     51    50    D0      063D  2793          MOVL    R0,R1                      ; Copy listhead address
                        0640  2794          ASSUME  SHR_L_QFL EQ 0
     51    61    D0      0640  2795 20$:     MOVL    (R1),R1                    ; Get next in list
     50    51    D1      0643  2796          CMPL    R1,R0                      ; Back to start of list?
           06    13      0646  2797          BEQL    30$                        ; Br if yes - no pid/chan match
           08    10      0648  2798          BSBB    CHECK_SHR                  ; Check for match
           F4    12      064A  2799          BNEQ    20$                        ; Br if none
           03    11      064C  2800          BRB     40$                        ; Return in success
     50    50    D0      064E  2801 30$:     MOVL    R0,R0                      ; Return match failure
           05    05      0651  2802 40$:     RSB
                        0652  2803
                        0652  2804 ;+
                        0652  2805 ; Subroutine to check if PID and SHR data base match up
                        0652  2806 ;
                        0652  2807 ; Inputs:
                        0652  2808 ;       R1 = Address of SHR
                        0652  2809 ;       R3 = Address of IRP
                        0652  2810 ;
                        0652  2811 ; Outputs:
                        0652  2812 ;       Z-Bit set then match.
                        0652  2813 ;       Z-Bit clear then no match.
                        0652  2814 ;-
                        0652  2815
                        0652  2816 CHECK_SHR:                                   ; Check for match with SHR data base
        0C A3    D5      0652  2817          TSTL    IRP$L_PID(R3)              ; Is this an Internal IRP user?
           09    19      0655  2818          BLSS    10$                        ; Br if yes, only one allowed per UCB
                        0657  2819 ;
                        0657  2820 ; Normal QIO user
                        0657  2821 ;
  0C A1  0C A3    D1      0657  2822          CMPL    IRP$L_PID(R3),SHR_L_PID(R1) ; PIDs match?
           0F    12      065C  2823          BNEQ    30$                        ; Br if no - try for next
           08    11      065E  2824          BRB     20$                        ; Else, continue checks
                        0660  2825 ;
                        0660  2826 ; Internal IRP user
                        0660  2827 ;
  0C A1  00B8 C5    D1      0660  2828 10$:     CMPL    UCB$L_XE_PID(R5),SHR_L_PID(R1) ; Is this the Internal user?
```

```
                05   12  0666  2829          BNEQ    30$                             ; Br if not
          10 A1 28 A3   B1  0668  2830 20$:    CMPW    IRP$W_CHAN(R3),SHR_W_CHAN(R1) ; Channels match?
                05      066D  2831 30$:    RSB                                     ; Return to caller
                        066E  2832
```

```
                     066E  2834          .SBTTL  ALT_START - ALTERNATE START I/O ROUTINE
                     066E  2835 ;++
                     066E  2836 ; ALT_START - ALTERNATE STARTIO I/O ROUTINE
                     066E  2837 ;
                     066E  2838 ; Functional description:
                     066E  2839 ;
                     066E  2840 ; This routine is called by the Executive to pass an "internal" IRP
                     066E  2841 ; to the driver.  "Internal" IRP's are those not built via QIO.
                     066E  2842 ; These IRPs are used by higher level software used to request I/O and
                     066E  2843 ; should not be confused with the IRPs built and passed by the
                     066E  2844 ; Transport layer to NSP.  The action here is to setup the IRP fields
                     066E  2845 ; as if the packet had been processed by the FDT routines.
                     066E  2846 ;
                     066E  2847 ; Inputs:
                     066E  2848 ;
                     066E  2849 ;       R3 = IRP address
                     066E  2850 ;       R5 = UCB address
                     066E  2851 ;
                     066E  2852 ;       All pertinent fields of the IRP are assumed to be valid.
                     066E  2853 ;
                     066E  2854 ;       IPL = FIPL
                     066E  2855 ;
                     066E  2856 ; Implicit inputs:
                     066E  2857 ;
                     066E  2858 ;       IRP$L_SVAPTE(R3) = System VIRTUAL address (not physical PTE address)
                     066E  2859 ;
                     066E  2860 ; Outputs:
                     066E  2861 ;
                     066E  2862 ;       R0-R5 may be garbage
                     066E  2863 ;--
                     066E  2864
                     066E  2865 ALT_START::                             ; Accept an "internal" IRP
          01   E0    066E  2866          BBS     #IRP$V_FUNC,-          ; If BS then read function
     2D 2A A3         0670  2867                  IRP$W_STS(R3),10$
      00CA C5   B0    0673  2868          MOVW    UCB$W_XE_PROTYP(R5),-  ; MUST be a non-promiscous user
        3A A3         0677  2869                  IRP$W_XE_PROTYP(R3)    ;
   52    2C A3   D0    0679  2870          MOVL    IRP$L_SVAPTE(R3),R2    ; Get address of start of data
51    62    0E   C3    067D  2871          SUBL3   #XE_C_HEADER,(R2),R1   ; Get the xmit buffer address
      51    52   C2    0681  2872          SUBL    R2,R1                  ; Form offset to start of data
   18 A2    51   B0    0684  2873          MOVW    R1,CXB$W_BOFF(R2)      ; Store offset in CXB
   0A A2    1B   90    0688  2874          MOVB    #DYN$C_CXB,CXB$B_TYPE(R2) ; Set structure type to CXB
      00D8 C5   91    068C  2875          CMPB    UCB$B_XE_PRO(R5),-     ;% Point-to-point mode?
               00         0690  2876                  #NMA$C_LINPR_POI
               03   12    0691  2877          BNEQ    5$                     ;% If so,
         40 A3   7C    0693  2878          CLRQ    IRP$Q_STATION(R3)      ;% Pick up destination from SHR block
               53   DD    0696  2879 5$:     PUSHL   R3                     ; Save IRP address
            FD26    30    0698  2880          BSBW    XMT_START              ; Start transmit operation
               53 8ED0    069B  2881          POPL    R3                     ; Restore IRP address
               04   11    069E  2882          BRB     30$                    ; Continue
                     06A0  2883
   52    2C A3   D0    06A0  2884 10$:    MOVL    IRP$L_SVAPTE(R3),R2    ; Get address of input buffer
               0B   13    06A4  2885          BEQL    20$                    ; Br if none
54    0180 C5   D0    06A6  2886          MOVL    UCB$L_XE_CDB(R5),R4    ; Get CDB address
         2C A3   D4    06AB  2887          CLRL    IRP$L_SVAPTE(R3)       ; Make sure SVAPTE is cleared
                     06AE  2888 ;
                     06AE  2889 ; The driver must be prepared to process chained buffers returned from
                     06AE  2890 ; the higher levels.
```

```
                     06AE  2891 ;
                     06AE  2892 ;&&15$: PUSHL   CXB$L_LINK(R2)          ; Save address of next in link
                     06AE  2893 ;&&     CLRL    CXB$L_LINK(R2)          ; Clear the link cell
 0CFE      30        06AE  2894         BSBW    ADDRCVLIST              ; Else, add it to the receive list
                     06B1  2895 ;&&     POPL    R2                      ; Get back address of next in chain
                     06B1  2896 ;&&     BNEQ    15$                     ; Br if more to return
                     06B1  2897
 FF33      30        06B1  2898 20$:    BSBW    RCV_START               ; Start receive operation
01 50      E9        06B4  2899 30$:    BLBC    R0,40$                  ; Br if error
           05        06B7  2900         RSB                             ; Return to caller
                     06B8  2901
 1890      31        06B8  2902 40$:    BRW     IO_DONE                 ; Post the I/O request in error
```

```
06BB  2904          .SBTTL  SETMODE_FDT - SET MODE I/O OPERATION FDT DISPATCH ROUTINE
06BB  2905 ;++
06BB  2906 ; SETMODE_FDT - SET MODE FDT PROCESSING
06BB  2907 ;
06BB  2908 ; Functional description:
06BB  2909 ;
06BB  2910 ; This is the fdt routine for setmode functions.
06BB  2911 ; There are three functions based on subfunction modifier bit.
06BB  2912 ;
06BB  2913 ; NOTE: That there is no difference on a request to shutdown a line or
06BB  2914 ;       a circuit. However, a request to startup a circuit is ignored
06BB  2915 ;       completely.
06BB  2916 ;
06BB  2917 ; The QIO parameters for SETMODE are:
06BB  2918 ;
06BB  2919 ;       P2 = Optional address of buffer descriptor for extended characteristics
06BB  2920 ;
06BB  2921 ;
06BB  2922 ; The Subfunction modifiers are as follows:
06BB  2923 ;
06BB  2924 ; 1)    CHANGE MODE -- NO MODIFIER BIT.
06BB  2925 ;       This function is done in the STARTIO routine. Control is passed to
06BB  2926 ;       EXE$SETMODE to validate the new mode buffer and queue the packet.
06BB  2927 ;
06BB  2928 ; 2)    INITIALIZE THE UNIT -- IO$M_STARTUP SET.
06BB  2929 ;       This function is done partially here and the remainder in STARTIO.
06BB  2930 ;       The action here is to pick up the user buffered I/O quota.  The quota
06BB  2931 ;       taken from the user is in IRP$W_BOFF. This value will be the IOSB+2 value
06BB  2932 ;       at I/O done. The mailbox is enabled and a receive is started.
06BB  2933 ;
06BB  2934 ; 3)    SHUTDOWN UNIT -- IO$M_SHUTDOWN SET.
06BB  2935 ;       This function shuts down the unit and optionally resets the mode.
06BB  2936 ;       A CANCEL I/O is performed, all outstanding I/O is completed, the
06BB  2937 ;       message blocks are all returned and the unit is left in an idle
06BB  2938 ;       state. This function cannot be done here and the FDT processing is
06BB  2939 ;       that of all SETMODE operations.
06BB  2940 ;
06BB  2941 ; 4)    ATTENTION AST -- IO$M_ATTNAST SET.
06BB  2942 ;       This function sets up an AST to be delivered when a change of
06BB  2943 ;       status occurs on the DEUNA/DELUA.
06BB  2944 ;
06BB  2945 ;
06BB  2946 ; Inputs:
06BB  2947 ;
06BB  2948 ;       R3 = IRP ADDESS
06BB  2949 ;       R4 = PCB ADDRESS
06BB  2950 ;       R5 = UCB ADDRESS
06BB  2951 ;       R6 = CCB ADDRESS
06BB  2952 ;       R7 = FUNCTION CODE
06BB  2953 ;       AP = ADDRESS OF THE FIRST QIO PARAMETER
06BB  2954 ;
06BB  2955 ;       IPL = ASTDEL
06BB  2956 ;
06BB  2957 ; Outputs:
06BB  2958 ;
06BB  2959 ;       R3 = IRP ADDRESS
06BB  2960 ;       R4 = PCB ADDRESS
```

```
                        06BB  2961 ;       R5 = UCB ADDRESS
                        06BB  2962 ;
                        06BB  2963 ;       R0-R2,R6 are destroyed.
                        06BB  2964 ;
                        06BB  2965 ;--
                        06BB  2966
                        06BB  2967 SETMODE_FDT::                            ; SET MODE FDT processing
          2C A3   D4    06BB  2968         CLRL    IRP$L_SVAPTE(R3)         ; Set no buffered packet
          38 A3   7C    06BE  2969         CLRQ    IRP$L_MEDIA(R3)          ; Reset mode data area
    50    0084 8F   3C    06C1  2970         MOVZWL  #SS$_DEVOFFLINE,R0       ; Assume unit if offline
               04   E0    06C6  2971         BBS     #UCB$V_ONLINE,-          ; Br if unit online
        03 64 A5          06C8  2972                 UCB$W_STS(R5),5$         ;
             FCB5   31    06CB  2973         BRW     ABORTIO                  ; Else, abort the I/O request
                        06CE  2974
      57    20 A3   B0    06CE  2975 5$:     MOVW    IRP$W_FUNC(R3),R7        ; Get entire function code
    5C 57    08   E1    06D2  2976         BBC     #IO$V_ATTNAST,R7,30$     ; Br if not attention AST
                        06D6  2977 ;
                        06D6  2978 ; User is requesting an attention AST
                        06D6  2979 ;
    57    00C0 C5   DE    06D6  2980         MOVAL   UCB$L_XE_AST(R5),R7      ; Get address of AST list
     00000000'GF    16    06DB  2981         JSB     G^COM$SETATTNAST         ; Set up attention AST
                        06E1  2982
                        06E1  2983         ASSUME  UCB$V_XE_INITED EQ 0
        12 68 A5   E9    06E1  2984         BLBC    UCB$W_DEVSTS(R5),10$     ; Br if protocol not active
                        06E5  2985
    51    00A0 C5   9E    06E5  2986         MOVAB   UCB$Q_XE_RCVMSG(R5),R1   ; Check for empty receive list
          61    51   D1    06EA  2987         CMPL    R1,(R1)                  ; Empty?
                08   13    06ED  2988         BEQL    10$                      ; Br if YES, no need to inform user
                53   DD    06EF  2989         PUSHL   R3                       ; Save IRP address
              29D6   30    06F1  2990         BSBW    POKE_USER                ; Inform user
                53 8ED0    06F4  2991         POPL    R3                       ; Restore IRP address
        50    01   9A    06F7  2992 10$:     MOVZBL  S^#SS$_NORMAL,R0         ; Set success
    51    44 A5   D0    06FA  2993         MOVL    UCB$L_DEVDEPEND(R5),R1   ; Get device dependent information
00000000'GF    17    06FE  2994 20$:     JMP     G^EXE$FINISHIO           ; Complete the I/O
                        0704  2995 ;
                        0704  2996 ; On a circuit request,
                        0704  2997 ;       If this is a shutdown then perform a $CANCEL and clear the RUN flag.
                        0704  2998 ;       If this is a startup then set the RUN flag.
                        0704  2999 ;
    14 57    07   E1    0704  3000 25$:     BBC     #IO$V_SHUTDOWN,R7,27$    ; Br if not a shutdown request
                        0708  3001         SETIPL  UCB$B_FIPL(R5)           ; Sync acces to UCB & CDB
                        070C  3002         CLRBIT  #UCB$V_XE_RUN,-          ; Clear the RUN flag
                        070C  3003                 UCB$W_DEVSTS(R5)
    52    28 A3   3C    0711  3004         MOVZWL  IRP$W_CHAN(R3),R2        ; Get channel number
                        0715  3005         ASSUME  CAN$C_CANCEL EQ 0
                58   D4    0715  3006         CLRL    R8                       ; Set $CANCEL function
              2187   30    0717  3007         BSBW    CANCEL                   ; Perform a CANCEL
                DB   11    071A  3008         BRB     10$                      ; Complete the request
                        071C  3009
    D7 57    06   E1    071C  3010 27$:     BBC     #IO$V_STARTUP,R7,10$     ; Br if not a startup request
                        0720  3011         CLRBIT  #XM$V_ERR_START,UCB$L_DEVDEPEND(R5) ;% Clear start error flag
        0060 8F   A8    0725  3012         BISW    #UCB$M_XE_START!UCB$M_XE_STACK,- ;% Set start and stack states
          68 A5         0729  3013                 UCB$W_DEVSTS(R5)         ;%
                        072B  3014         SETBIT  #UCB$V_XE_RUN,-          ; Set the RUN flag
                        072B  3015                 UCB$W_DEVSTS(R5)
                C5   11    0730  3016         BRB     10$                      ; Complete the request
                        0732  3017 ;
```

```
                     0732  3018 ; For everthing except Attention ASTs we must make sure CDB is present
                     0732  3019 ; and we must verify the P2 buffer.
                     0732  3020 ;
      0180 C5   D5   0732  3021 30$:    TSTL    UCB$L_XE_CDB(R5)          ; Is CDB there?
           0E   12   0736  3022          BNEQ    35$                       ; Br if yes
         1C33   30   0738  3023          BSBW    ALLOC_CDB                 ; Else, allocate CDB
        08 50   E8   073B  3024          BLBS    R0,35$                    ; Br if successful
50    0124 8F   3C   073E  3025          MOVZWL  #SS$_INSFMEM,R0           ; Set error return
         FC3D   31   0743  3026 33$:    BRW     ABORTIO                   ; Return error
                     0746  3027
         0649   30   0746  3028 35$:    BSBW    GET_CHAR_BUF              ; Get P2 characteristics
        F7 50   E9   0749  3029          BLBC    R0,33$                    ; Br if error - Abort I/O
52    F936 CF   9E   074C  3030          MOVAB   LINE_PARAM_W0,R2          ; Assume the line parameters
  05 57   09   E0   0751  3031          BBS     #IO$V_CTRL,R7,36$         ; Br if line request
52    F9C5 CF   9E   0755  3032          MOVAB   CIRCUIT_PARAM,R2          ; Else, use the circuit parameters
         241E   30   075A  3033 36$:    BSBW    VALIDATE_P2               ; Validate the P2 parameters
        9E 50   E9   075D  3034          BLBC    R0,20$                    ; Br if error
  A0 57   09   E1   0760  3035          BBC     #IO$V_CTRL,R7,25$         ; Br if not a LINE request
                     0764  3036          SETIPL  UCB$B_FIPL(R5)            ; Sync access to UCB's
         23E8   30   0768  3037          BSBW    SAV_MULTI                 ; Save the multicast address list
      0090 C3   D4   076B  3038          CLRL    IRP$L_XE_SHR(R3)          ;  Assume exclusive user
           03   E1   076F  3039          BBC     #UCB$V_XE_SHARE,-         ; Br if not a SHARED user
     16 68 A5         0771  3040                  UCB$W_DEVSTS(R5),40$      ;
                     0774  3041 ;
                     0774  3042 ;       Allow the shared user to change the destination node with which
                     0774  3043 ;       it is communicating.
                     0774  3044 ;
         FEB6   30   0774  3045          BSBW    MATCH_SHR                 ; Else, try to find shared user
           11   12   0777  3046          BNEQ    40$                       ; Br if none found, skip it
0090 C3    51   D0   0779  3047          MOVL    R1,IRP$L_XE_SHR(R3)       ; Save the SHR data structure address
      12 A1   D0   077E  3048          MOVL    SHR_G_DEST(R1),-          ; Save the current destination user
      00CC C5         0781  3049                  UCB$G_XE_DES(R5)          ;  address in the UCB
        16 A1   B0   0784  3050          MOVW    SHR_G_DEST+4(R1),-        ;       ...
      00D0 C5         0787  3051                  UCB$G_XE_DES+4(R5)        ;       ...
                     078A  3052 ;
                     078A  3053 ;       Now we will set the parameters given in the setmode request. But,
                     078A  3054 ;       first if the DEUNA/DELUA controller is inited we will use the current
                     078A  3055 ;       hardware settings for the defaults.
                     078A  3056 ;
51    0180 C5   D0   078A  3057 40$:    MOVL    UCB$L_XE_CDB(R5),R1       ; Get CDB address
                     078F  3058          ASSUME  CDB_STS_V_INITED EQ 0
  13 025A C1   E9   078F  3059          BLBC    CDB_B_STS(R1),45$         ; Br if controller not enabled, use
                     0794  3060                                            ;  the fixed defaults
51    025F C1   9E   0794  3061          MOVAB   CDB_B_SETPRM(R1),R1       ; Get address of setable parameters
52    00DD C5   9E   0799  3062          MOVAB   UCB$B_XE_CDBPRM(R5),R2    ; Get address of UCB parameters
      50   04   9A   079E  3063          MOVZBL  #UCB$C_XE_CDBPRM,R0       ; Get size of parameters to move
      82   81   90   07A1  3064 42$:    MOVB    (R1)+,(R2)+               ; Store CDB parameters into UCB
        FA 50   F5   07A4  3065          SOBGTR  R0,42$                    ; Loop if more
52    F8DB  F   9E   07A7  3066 45$:    MOVAB   LINE_PARAM_W0,R2          ; Get address of verification table
         2538   30   07AC  3067          BSBW    CHANGE_PARAM              ; Change the parameters
51    0090 C3   D0   07AF  3068          MOVL    IRP$L_XE_SHR(R3),R1       ; Get the SHR structure address
           0C   13   07B4  3069          BEQL    50$                       ; Br if not present, skip it
      00CC C5   D0   07B6  3070          MOVL    UCB$G_XE_DES(R5),-        ; Else, reset the destination user
        12 A1         07BA  3071                  SHR_G_DEST(R1)            ;  address into the SHR structure
      00D0 C5   B0   07BC  3072          MOVW    UCB$G_XE_DES+4(R5),-      ;       ...
        16 A1         07C0  3073                  SHR_G_DEST+4(R1)          ;       ...
                     07C2  3074
```

```
                                07C2  3075 50$:    BBC     #IO$V_SHUTDOWN,R7,60$   ; Br if not shutdown request
                                07C6  3076 ;
                                07C6  3077 ; Shutdown protocol request
                                07C6  3078 ;
                                07C6  3079         ASSUME  UCB$V_XE_INITED  EQ  0
                  03 68 A5  E8  07C6  3080         BLBS    UCB$W_DEVSTS(R5),55$    ; Br if still inited
                      FF2A  31  07CA  3081         BRW     10$                     ; Else, complete I/O request now
                21 A3   09  90  07CD  3082 55$:    MOVB    S^#XE_FC_V_STOP,IRP$B_XE_FUNC(R3) ; Set internal function code
                      0424  31  07D1  3083         BRW     QUEPKT                  ; Queue request to DEUNA/DELUA
                                07D4  3084
                03 57   06  E0  07D4  3085 60$:    BBS     #IO$V_STARTUP,R7,80$    ; Br if startup function
                      0102  31  07D8  3086 70$:    BRW     180$                    ; Else, must be change mode
                                07DB  3087 ;
                                07DB  3088 ; Startup protocol request
                                07DB  3089 ;
                21 A3   00  90  07DB  3090 80$:    MOVB    #XE_FC_V_INIT,IRP$B_XE_FUNC(R3) ; Insert internal function code
                                07DF  3091 ;
                                07DF  3092 ;       If the UCB is already initialized for SHARED use, then we will
                                07DF  3093 ;       check to make sure that the SHR struture exists. If it does then
                                07DF  3094 ;       the share structure must be active, by definition.
                                07DF  3095 ;
                  0090 C3   D5  07DF  3096         TSTL    IRP$L_XE_SHR(R3)                ; Was the SHR structure present
                       F3   12  07E3  3097         BNEQ    70$                     ; Br if yes, already started
                       03   E0  07E5  3098         BBS     #UCB$V_XE_SHARE,-       ; Br if SHARED UCB, ingore status
                61 68 A5        07E7  3099                 UCB$W_DEVSTS(R5),125$   ; ..make user a shared user
                                07EA  3100         ASSUME  UCB$V_XE_INITED  EQ  0
                EA 68 A5  E8    07EA  3101         BLBS    UCB$W_DEVSTS(R5),70$    ; Br if already started
                                07EE  3102                                         ; ..check multicast address list
                       02   E1  07EE  3103         BBC     #UCB$V_XE_PROTYP,-      ; Br if no protocol specified
                43 68 A5        07F0  3104                 UCB$W_DEVSTS(R5),100$   ; ..error
                                07F3  3105 ;
                                07F3  3106 ; Check if protocol type is to be shared
                                07F3  3107 ;
                          03    91  07F3  3108         CMPB    #NMA$C_ACC_EXC,-        ; Is this PROTOCOL TYPE for exclusive
                      00D4 C5       07F5  3109                  UCB$B_XE_ACC(R5)       ;  use?
                           51   12  07F8  3110         BNEQ    125$                    ; Br if not
                                07FA  3111 ;
                                07FA  3112 ; Check protocol type for uniqueness
                                07FA  3113 ;
                     51   00CA C5   3C  07FA  3114         MOVZWL  UCB$W_XE_PROTYP(R5),R1  ; Get protocol type
                                07FF  3115 ;
                                07FF  3116 ; For a user wishing to run in promiscuous mode, the requirement is that
                                07FF  3117 ; there be no other promiscuous users running. For a non-promiscuous user,
                                07FF  3118 ; there must be no other users running with the same protocol type.
                                07FF  3119 ;
           50   000030FF'EF   9E  07FF  3120         MOVAB   MATCH_PROTYP,R0         ; Get address of Action routine
                                0806  3121                                         ;  assume non-PROMISCUOUS user
                                0806  3122
                                0806  3123         ASSUME  NMA$C_STATE_ON EQ 0
                                0806  3124         ASSUME  NMA$C_STATE_OFF EQ 1
                                0806  3125
                    1B 00DB C5  E8  0806  3126         BLBS    UCB$B_XE_PRM(R5),90$    ; Br if NOT a PROMISCUOUS user
                                080B  3127 ;
                                080B  3128 ; The promiscuous user must have PHY_IO privilege
                                080B  3129 ;
           50   00003121'EF   9E  080B  3130         MOVAB   MATCH_PROMTYP,R0        ; Get address of Action routine
                                0812  3131         IFPRIV  PHY_IO,90$              ; If user has privilege, then okay
```

```
                 234C   30  0818  3132          BSBW    RES_MULTI                       ; Else, Restore original multicast list
            50    24   3C  081B  3133          MOVZWL  #SS$_NOPRIV,R0                  ; Return error - NOPRIV
      51    0B18 8F   3C  081E  3134          MOVZWL  #NMA$C_PCLI_PRM,R1              ; Return the bad parameter
                 FED8   31  0823  3135          BRW     20$                             ; Finish the I/O request
                        0826  3136 90$:        PUSHQ   R4                              ; Save PCB, UCB addresses
      54    0180 C5   D0  0829  3137          MOVL    UCB$L_XE_CDB(R5),R4             ; Get CDB address
                   60   16  082E  3138          JSB     (R0)                            ; Try to find exact match
                        0830  3139          POPQ    R4                              ; Restore PCB, UCB addresses
                  22 50   E9  0833  3140          BLBC    R0,130$                         ; Br if none found - okay
                        0836  3141 ;
                        0836  3142 ;       Bad protocol type
                        0836  3143 ;
      51    0B0E 8F   3C  0836  3144 100$:  MOVZWL  #NMA$C_PCLI_PTY,R1              ; Return bad parameter code
                   05   11  083B  3145          BRB     120$                            ; Finish error reporting
                        083D  3146 ;
                        083D  3147 ;       Bad quota calculated
                        083D  3148 ;
      51    0451 8F   3C  083D  3149 110$:  MOVZWL  #NMA$C_PCLI_BFN,R1              ; Return bad parameter code
            50    14   9A  0842  3150 120$:  MOVZBL  S^#SS$_BADPARAM,R0             ; Set error return
                 231F   30  0845  3151 123$:  BSBW    RES_MULTI                       ; Restore original multicast list
                 FEB3   31  0848  3152          BRW     20$                             ; Finish the I/O request
                        084B  3153 ;
                        084B  3154 ; Shared protocol type - look for same protocol type in other UCB.
                        084B  3155 ;
                   0139 30  084B  3156 125$:  BSBW    SHR_UCB                         ; Find other UCB in user or make this
                        084E  3157                                                     ;  UCB shareable
                  F4 50   E9  084E  3158          BLBC    R0,123$                         ; Br on error
                        0851  3159 ;
                        0851  3160 ; We must now check if UCB is already inited.
                        0851  3161 ;
                        0851  3162          ASSUME  UCB$V_XE_INITED  EQ  0
                11 68 A5   E9  0851  3163          BLBC    UCB$W_DEVSTS(R5),135$           ; Br if this UCB is NOT inited
                        0855  3164                                                     ; skip quota taking, already done
                     008C   31  0855  3165          BRW     185$                            ; Else, compute multicast list
                        0858  3166
                        0858  3167 130$:  SETIPL  #IPL$_ASTDEL                    ; Reset IPL to ASTDEL
                        085B  3168 ;
                        085B  3169 ; Take quota needed
                        085B  3170 ;
                     0097   30  085B  3171          BSBW    TAKE_QUOTA                      ; Take quota from user
                   E4 50   E9  085E  3172          BLBC    R0,123$                         ; Br if error
      00C8 C5      57   B0  0861  3173          MOVW    R7,UCB$W_XE_QUOTA(R5)           ; Save quota in UCB
                        0866  3174 135$:  SETBIT  #UCB$V_XE_INITED,-              ; Indicate unit is initialized
                        0866  3175                UCB$W_DEVSTS(R5)                ;       ...
                        086B  3176          SETIPL  UCB$B_FIPL(R5)                  ; Sync access to UCB and CDB
        0A 00DB C5   E8  086F  3177          BLBS    UCB$B_XE_PRM(R5),140$           ; Br if NOT a PROMISCUOUS user
      54    0180 C5   D0  0874  3178          MOVL    UCB$L_XE_CDB(R5),R4             ; Get CDB address
      0224 C4      55   D0  0879  3179          MOVL    R5,CDB_L_PRMUSER(R4)            ; Store promiscuous user's address
                        087E  3180 ;
                        087E  3181 ; Pre-allocate all needed receive buffers, if the CDB is not initialized yet!
                        087E  3182 ; The buffers are immediately deallocated, but this pre-allocation will allow
                        087E  3183 ; the pool to grow if necessary! This must be done here, before we run on the
                        087E  3184 ; interrupt stack.
                        087E  3185 ;
                        087E  3186 140$:  PUSHQ   R3                              ; Save R3, R4
      51    00D6 C5   3C  0881  3187          MOVZWL  UCB$W_XE_BSZ(R5),R1             ; Get size of receive buffer
      54    00D2 C5   3C  0886  3188          MOVZWL  UCB$W_XE_HBQ(R5),R4             ; Get device buffer quota
```

```
                54    51    C6  088B  3189          DIVL    R1,R4                           ; Compute number of buffers to allocate
                      54    D6  088E  3190          INCL    R4                              ; Plus one extra
                      7E    D4  0890  3191          CLRL    -(SP)                           ; End of list marker
          00000000'GF       16  0892  3192 145$:     JSB     G^EXE$ALONONPAGED               ; Allocate the memory
                   0E   50    E9  0898  3193          BLBC    R0,150$                         ; Br on error
             08    A2   51    B0  089B  3194          MOVW    R1,IRP$W_SIZE(R2)               ; Save size of buffer
             0A    A2   1B    90  089F  3195          MOVB    #DYN$C_CXB,IRP$B_TYPE(R2)       ; Set structure type
                7E    52    D0  08A3  3196          MOVL    R2,-(SP)                        ; Save buffer address
                   E9   54    F5  08A6  3197          SOBGTR  R4,145$                         ; Loop if more to allocate
                               08A9  3198
                50    8E    D0  08A9  3199 150$:     MOVL    (SP)+,R0                        ; Get buffer address
                      08    13  08AC  3200          BEQL    155$                            ; Br if end of list
          00000000'GF       16  08AE  3201          JSB     G^EXE$DEANONPAGED               ; Deallocate the block
                      F3    11  08B4  3202          BRB     150$                            ; Try for more
                               08B6  3203 155$:     POPQ    R3                              ; Restore R3, R4
                               08B9  3204
                    2211     30  08B9  3205          BSBW    ADD_MULTI                       ; Compile a new multicast address list
                   33 50     E8  08BC  3206          BLBS    R0,T90$                         ; Br if all okay
                               08BF  3207
                               08BF  3208          ASSUME  NMA$C_STATE_ON EQ 0
                               08BF  3209          ASSUME  NMA$C_STATE_OFF EQ 1
                               08BF  3210
             04 00DB C5        E8  08BF  3211          BLBS    UCB$B_XE_PRM(R5),160$           ; Br if NOT a PROMISCUOUS user
                0224 C4        D4  08C4  3212          CLRL    CDB_L_PRMUSER(R4)               ; Else, clear the PROMISCUOUS user addr
                               08C8  3213 160$:     CLRBIT  #UCB$V_XE_INITED,-              ; Indicate unit is not initialized
                               08C8  3214                  UCB$W_DEVSTS(R5)            ;               ...
             20    A6   57    C0  08CD  3215          ADDL    R7,JIB$L_BYTCNT(R6)             ; Restore quota
             24    A6   57    C0  08D1  3216          ADDL    R7,JIB$L_BYTLM(R6)              ; ..and byte limit
          51    0B0F   8F    3C  08D5  3217 170$:     MOVZWL  #NMA$C_PCLI_MCA,R1              ; Indicate bad multicast address
                   FF65    31  08DA  3218          BRW     120$                            ; Return in error
                               08DD  3219 ;
                               08DD  3220 ; Change mode request - might have to reset multicast address list
                               08DD  3221 ;
                               08DD  3222 180$:     ASSUME  UCB$V_XE_INITED  EQ  0
                03 68 A5       E8  08DD  3223          BLBS    UCB$W_DEVSTS(R5),185$           ; Br if unit inited
                   FE13        31  08E1  3224          BRW     10$                             ; Else, success
                               08E4  3225
                               08E4  3226 185$:     SETIPL  UCB$B_FIPL(R5)                  ; Sync access to UCB and CDB
                    21E2       30  08E8  3227          BSBW    ADD_MULTI                       ; Compile global multicast address list
                  E7 50        E9  08EB  3228          BLBC    R0,T70$                         ; Br if error
             21 A3    0C       90  08EE  3229          MOVB    #XE_FC_V_MULTI,IRP$B_XE_FUNC(R3) ; Set function request
                   0303        31  08F2  3230 190$:     BRW     QUEPKT                          ; Queue packet to driver
```

```
                     08F5  3232 ;+
                     08F5  3233 ; Take quota subroutine
                     08F5  3234 ;
                     08F5  3235 ;       Calculate buffer quota and check against user's quota
                     08F5  3236 ;
                     08F5  3237 ; Inputs:       R7 = Scratch
                     08F5  3238 ;               R6 = Scratch
                     08F5  3239 ;               R5 = UCB address
                     08F5  3240 ;               R4 = PCB address
                     08F5  3241 ;
                     08F5  3242 ;
                     08F5  3243 ; Outputs:      R7 = Quota taken
                     08F5  3244 ;               R6 = JIB address
                     08F5  3245 ;               R5 = UCB address
                     08F5  3246 ;               R4 = PCB address
                     08F5  3247 ;               R0 = Status
                     08F5  3248 ;
                     08F5  3249 ;       R1,R2 are destroyed.
                     08F5  3250 ;
                     08F5  3251 ; Implicit outputs:
                     08F5  3252 ;
                     08F5  3253 ;       BR to ABORTIO if quota is exceeded.
                     08F5  3254 ;
                     08F5  3255 ;-
                     08F5  3256 TAKE_QUOTA:
          50   D4    08F5  3257         CLRL    R0                          ; Assume failure
      00D6 C5   B6    08F7  3258         INCW    UCB$W_XE_BSZ(R5)            ; Round buffer size to even value
00D6 C5   01   AA    08FB  3259         BICW    #1,UCB$W_XE_BSZ(R5)         ;       ...
51    00D5 C5   9A    0900  3260         MOVZBL  UCB$B_XE_BFN(R5),R1         ; Get number of receive buffers
  52    42 A5   3C    0905  3261         MOVZWL  UCB$W_DEVBUFSIZ(R5),R2      ; Get buffer size
     51   52   C4    0909  3262         MULL    R2,R1                       ; Get needed quota
     57   51   3C    090C  3263         MOVZWL  R1,R7                       ; Copy quota
     57   51   D1    090F  3264         CMPL    R1,R7                       ; Overflow?
          30   12    0912  3265         BNEQ    110$                        ; Br if yes - error
  7E    24 A4   D0    0914  3266         MOVL    PCB$L_STS(R4),-(SP)         ; Save current PCB status
                     0918  3267         SETBIT  #PCB$V_SSRWAIT,-            ; Do not go into a resource wait
                     0918  3268                 PCB$L_STS(R4)               ;   just to check the quota
          53   DD    091D  3269         PUSHL   R3                          ; Save R3
00000000'GF    16    091F  3270         JSB     G^EXE$BUFQUOPRC             ; Check quota
          53 8ED0    0925  3271         POPL    R3                          ; Restore R3
  24 A4    8E   D0    0928  3272         MOVL    (SP)+,PCB$L_STS(R4)         ; Restore previous PCB status
        02 50   E8    092C  3273         BLBS    R0,50$                      ; Br if success
          10   11    092F  3274         BRB     80$                         ; Else, return error
                     0931  3275
56    0080 C4   D0    0931  3276 50$:    MOVL    PCB$L_JIB(R4),R6            ; Get JIB address
  20 A6    57   C2    0936  3277         SUBL    R7,JIB$L_BYTCNT(R6)         ; Adjust quota
  24 A6    57   C2    093A  3278         SUBL    R7,JIB$L_BYTLM(R6)          ; ..and byte limit
     50    01   9A    093E  3279         MOVZBL  #1,R0                       ; Indicate success
           51   D4    0941  3280 80$:    CLRL    R1                          ; No error return
                05    0943  3281 90$:    RSB
                     0944  3282
     50    14   9A    0944  3283 110$:   MOVZBL  #SS$_BADPARAM,R0            ; Setup error code
51   0451 8F    3C    0947  3284         MOVZWL  #NMA$C_PCLI_BFN,R1          ; Assume BAD BFN
           F5   11    094C  3285         BRB     90$                         ; Return error
                     094E  3286
                     094E  3287 ;+
                     094E  3288 ; CHECK_QUOTA - check SHARED unit's quota
```

```
                                094E  3289 ;
                                094E  3290 ; Inputs:       R9 = Scratch
                                094E  3291 ;               R7 = Original UCB address
                                094E  3292 ;               R5 = UCB address
                                094E  3293 ;               R4 = PCB address
                                094E  3294 ;
                                094E  3295 ;
                                094E  3296 ; Outputs:      R9 = Quota taken
                                094E  3297 ;               R7 = Original UCB address
                                094E  3298 ;               R5 = UCB address
                                094E  3299 ;               R4 = PCB address
                                094E  3300 ;               R0 = Status
                                094E  3301 ;
                                094E  3302 ;       R1,R2 are destroyed.
                                094E  3303 ;
                                094E  3304 ;-
                                094E  3305 CHECK_QUOTA:
                                094E  3306          PUSHQ   R6                          ; Save R6, R7
                          55    DD  0951  3307          PUSHL   R5                          ; Save UCB address
                     55   57    D0  0953  3308          MOVL    R7,R5                       ; Copy original UCB address
                          9D    10  0956  3309          BSBB    TAKE_QUOTA                  ; Charge quota to user
                          55 8ED0   0958  3310          POPL    R5                          ; Restore UCB address
                        16 50    E9  095B  3311          BLBC    R0,90$                      ; Br if error
                     59   57    D0  095E  3312          MOVL    R7,R9                       ; Copy quota taken
                51   0175 C5    3C  0961  3313          MOVZWL  UCB$W_XE_TOTQUO(R5),R1      ; Get total quota
                     51   59    C0  0966  3314          ADDL    R9,R1                       ; Compute new total
                     50   51    3C  0969  3315          MOVZWL  R1,R0                       ; Copy quota
                     50   51    D1  096C  3316          CMPL    R1,R0                       ; Overflow?
                          07    12  096F  3317          BNEQ    110$                        ; Br if yes, error
                     50   01    9A  0971  3318          MOVZBL  #1,R0                       ; Else, return success
                                    0974  3319 90$:      POPQ    R6                          ; Retore R6, R7
                                05  0977  3320          RSB                                 ; Return to caller
                                    0978  3321
              20 A6      59    C0  0978  3322 110$:     ADDL    R9,JIB$L_BYTCNT(R6)         ; Restore quota
              24 A6      59    C0  097C  3323           ADDL    R9,JIB$L_BYTLM(R6)          ; ..and byte limit
                 50      1C    3C  0980  3324           MOVZWL  #SS$_EXQUOTA,R0             ; Return bad quota
                         51    D4  0983  3325           CLRL    R1                          ; No parameter return
                         ED    11  0985  3326           BRB     90$                         ; Exit with error
                                    0987  3327
```

```
                                0987  3329          .SBTTL  SHR_UCB - CREATE SHARED UCB
                                0987  3330 ;++
                                0987  3331 ; SHR_UCB - CREATE SHARED UCB
                                0987  3332 ;
                                0987  3333 ; Functional description:
                                0987  3334 ;
                                0987  3335 ; This subroutine creates a shared UCB if this is the first SHARED user of the
                                0987  3336 ; particular protocol type.  Else, the already created SHARED UCB is found and
                                0987  3337 ; a shared data structure is added to the list of shared users of that protocol
                                0987  3338 ; type.
                                0987  3339 ;
                                0987  3340 ; Inputs:
                                0987  3341 ;
                                0987  3342 ;       R3 = IRP address
                                0987  3343 ;       R4 = PCB address
                                0987  3344 ;       R5 = UCB address
                                0987  3345 ;       R6 = CCB address
                                0987  3346 ;       R7 = Function code
                                0987  3347 ;       AP = Address of first function-dependent QIO parameter
                                0987  3348 ;
                                0987  3349 ;       IPL = FIPL
                                0987  3350 ;
                                0987  3351 ; Outputs:
                                0987  3352 ;
                                0987  3353 ;       R0 = Status return for request
                                0987  3354 ;       R1 = Bad parameter code (if bad parameter error code)
                                0987  3355 ;       R2,R7 are destroyed.
                                0987  3356 ;       All other registers are preserved.
                                0987  3357 ;
                                0987  3358 ;       IPL = ASTDEL
                                0987  3359 ;
                                0987  3360 ;--
                                0987  3361
                                0987  3362 SHR_UCB::                                ; Setup shared protocol UCB
     0300 8F    BB              0987  3363          PUSHR   #^M<R8,R9>              ; Save registers
 00000000'GF    16              098B  3364          JSB     G^SCH$IOLOCKW           ; Lock I/O data base for write access
     57    55   D0              0991  3365          MOVL    R5,R7                   ; Save UCB address
50   21C4 8F    3C              0994  3366          MOVZWL  #SS$_DUPUNIT,R0         ; Assume 2 channels assigned to UCB
  5C A5    01   B1              0999  3367          CMPW    #1,UCB$W_REFC(R5)       ; Is there only one reference to UCB?
           72   12              099D  3368          BNEQ    23$                     ; Br if not, error (more than 1 channel)
58   00D4 C5    9A              099F  3369          MOVZBL  UCB$B_XE_ACC(R5),R8     ; Save access mode
           03   E0              09A4  3370          BBS     #UCB$V_XE_SHARE,-       ; Br if we are already the SHARED UCB
     29 68 A5                   09A6  3371                  UCB$W_DEVSTS(R5),10$    ;
           54   DD              09A9  3372          PUSHL   R4                      ; Save PCB address
54   0180 C5    D0              09AB  3373          MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
51   00CA C5    B0              09B0  3374          MOVW    UCB$W_XE_PROTYP(R5),R1  ; Get protocol type
        2747    30              09B5  3375          BSBW    MATCH_PROTYP            ; Try to match protocol type
           54 8ED0              09B8  3376          POPL    R4                      ; Restore PCB address
        14 50   E8              09BB  3377          BLBS    R0,10$                  ; Br if found
     55    57   D0              09BE  3378          MOVL    R7,R5                   ; Else, get back old UCB address
       2C A5    D4              09C1  3379          CLRL    UCB$L_PID(R5)           ; Make this UCB shareable
  50   1C A5    D0              09C4  3380          MOVL    UCB$L_ORB(R5),R0        ; Get the ORB address
           60   D4              09C8  3381          CLRL    ORB$L_OWNER(R0)         ;  clear owner UIC as well
     00C4 C5    D4              09CA  3382          CLRL    UCB$L_XE_DEFUSR(R5)     ; Clear default user
           08   A8              09CE  3383          BISW    #UCB$M_XE_SHARE,-       ; Indicate that UCB is in SHARED mode
       68 A5                    09D0  3384                  UCB$W_DEVSTS(R5)        ;
                                09D2  3385 ;
```

```
                        09D2  3386 ; Allocate a share data structure and link it in
                        09D2  3387 ;
      58    02    91    09D2  3388 10$:     CMPB    #NMA$C_ACC_LIM,R8       ; Is the new user a limited user?
            0B    13    09D5  3389          BEQL    20$                     ; Br if yes - okay
50    0840 8F     3C    09D7  3390          MOVZWL  #SS$_DEVALLOC,R0        ; Assume protocol already allocated
      00C4 C5     D5    09DC  3391          TSTL    UCB$L_XE_DEFUSR(R5)     ; Is there already a default user?
            2F    12    09E0  3392          BNEQ    23$                     ; Br if YES - error
         00C8     30    09E2  3393 20$:     BSBW    CHECK_PARAM             ; Check out all parameters
         2B 50    E9    09E5  3394          BLBC    R0,25$                  ; Br on error
          FF63    30    09E8  3395          BSBW    CHECK_QUOTA             ; Check our quota
         25 50    E9    09EB  3396          BLBC    R0,25$                  ; Br if error
      51    2A    9A    09EE  3397          MOVZBL  #SHR_C_LENGTH,R1        ; Get size of structure to allocate
            53    DD    09F1  3398          PUSHL   R3                      ; Save IRP address
 00000000'GF      16    09F3  3399          JSB     G^EXE$ALLOCBUF          ; Allocate buffer, reset IPL to ASTDEL
            53 8ED0     09F9  3400          POPL    R3                      ; Restore IRP address
         1A 50    E8    09FC  3401          BLBS    R0,30$                  ; Br if success
50    0124 8F     3C    09FF  3402          MOVZWL  #SS$_INSFMEM,R0         ; Else, return error reason
51    0080 C4     D0    0A04  3403          MOVL    PCB$L_JIB(R4),R1        ; Get JIB address
   20 A1    59    C0    0A09  3404          ADDL    R9,JIB$L_BYTCNT(R1)     ; Restore quota
   24 A1    59    C0    0A0D  3405          ADDL    R9,JIB$L_BYTLM(R1)      ; ..and byte limit
            51    D4    0A11  3406 23$:     CLRL    R1                      ; No bad parameter code
      55    57    D0    0A13  3407 25$:     MOVL    R7,R5                   ; Get back the OLD UCB address
         0085    31    0A16  3408          BRW     80$                     ; Exit with error
                        0A19  3409 ;
                        0A19  3410 ; Initialize shared (SHR) data structure
                        0A19  3411 ;
                        0A19  3412 30$:     ASSUME  SHR_L_QFL EQ 0
                        0A19  3413          ASSUME  SHR_L_QBL EQ SHR_L_QFL+4
            82    7C    0A19  3414          CLRQ    (R2)+                   ; Zero LINK pointers
                        0A1B  3415          ASSUME  SHR_W_SIZE EQ SHR_L_QBL+4
      82    51    B0    0A1B  3416          MOVW    R1,(R2)+                ; Save size
                        0A1E  3417          ASSUME  SHR_B_TYPE EQ SHR_W_SIZE+2
            52    D6    0A1E  3418          INCL    R2                      ; Filled by EXE$ALLOCBUF routine
                        0A20  3419          ASSUME  SHR_B_STS EQ SHR_B_TYPE+1
      82    01    90    0A20  3420          MOVB    #SHR_STS_M_INITED,(R2)+ ; Initialize SHR status
                        0A23  3421          ASSUME  SHR_L_PID EQ SHR_B_STS+1
   82    0C A3    D0    0A23  3422          MOVL    IRP$L_PID(R3),(R2)+     ; Save users PID and CHAN for
                        0A27  3423          ASSUME  SHR_W_CHAN EQ SHR_L_PID+4
   82    28 A3    B0    0A27  3424          MOVW    IRP$W_CHAN(R3),(R2)+    ;  for future lookups
                        0A2B  3425          ASSUME  SHR_G_DEST EQ SHR_W_CHAN+2
82    00CC C7     D0    0A2B  3426          MOVL    UCB$G_XE_DES(R7),(R2)+  ; Save destination address
82    00D0 C7     B0    0A30  3427          MOVW    UCB$G_XE_DES+4(R7),(R2)+ ;
                        0A35  3428          ASSUME  SHR_Q_QUEUES EQ SHR_G_DEST+6
      51    02    9A    0A35  3429          MOVZBL  #SHR_C_QUEUES,R1        ; Get number of queues in structure
      82    62    DE    0A38  3430 40$:     MOVAL   (R2),(R2)+              ; Set forward link pointer
   82    FC A2    D0    0A3B  3431          MOVL    -4(R2),(R2)+            ; Set backward link pointer
         F6 51    F5    0A3F  3432          SOBGTR  R1,40$                  ; Loop if more listheads
                        0A42  3433          ASSUME  SHR_W_QUOTA EQ SHR_Q_QUEUES+<8*SHR_C_QUEUES>
      82    59    B0    0A42  3434          MOVW    R9,(R2)+                ; Initialize quota
                        0A45  3435
                        0A45  3436          ASSUME  UCB$V_XE_INITED EQ 0
      0C 68 A5    E9    0A45  3437          BLBC    UCB$W_DEVSTS(R5),50$    ; Br if UCB not initialized
00C8 C5     59    A0    0A49  3438          ADDW    R9,UCB$W_XE_QUOTA(R5)   ; Add to the current quota
0175 C5     59    A0    0A4E  3439          ADDW    R9,UCB$W_XE_TOTQUO(R5)  ;  and the total quota
            0A    11    0A53  3440          BRB     55$                     ; Continue
00C8 C5     59    B0    0A55  3441 50$:     MOVW    R9,UCB$W_XE_QUOTA(R5)   ; Set current quota
0175 C5     59    B0    0A5A  3442          MOVW    R9,UCB$W_XE_TOTQUO(R5)  ;  and total quota
```

```
                52   2A   C2  0A5F  3443 55$:    SUBL    #SHR_C_LENGTH,R2          ; Backup to beginning of structure
                58   02   91  0A62  3444         CMPB    #NMA$C_ACC_LIM,R8         ; Is this for limited use?
                     07   13  0A65  3445         BEQL    60$                       ; Br if YES
          00C4  C5   52   D0  0A67  3446         MOVL    R2,UCB$L_XE_DEFUSR(R5)    ; Else, save default user address
                     05   11  0A6C  3447         BRB     65$                       ; Skip linking onto list
          009C  D5   62   0E  0A6E  3448 60$:    INSQUE  (R2),@UCB$Q_XE_SHARE+4(R5) ; Link user into shared user list
                5C   A5   B6  0A73  3449 65$:    INCW    UCB$W_REFC(R5)            ; Increment the ref count on the
                          0A76  3450                                               ;  UCB to be used
                57   55   D1  0A76  3451         CMPL    R5,R7                     ; Was this the original UCB?
                     20   13  0A79  3452         BEQL    70$                       ; Br if YES - no more work to do
                5C   A5   B6  0A7B  3453         INCW    UCB$W_REFC(R5)            ; Else, increment REFC (for $DASSGN)
             1C  A3   55   D0  0A7E  3454         MOVL    R5,IRP$L_UCB(R3)          ; Return new UCB address
                66   55   D0  0A82  3455         MOVL    R5,CCB$L_UCB(R6)          ;  in CCB also.
                     28   BB  0A85  3456         PUSHR   #^M<R3,R5>                ; Save IRP, real UCB address
                55   57   D0  0A87  3457         MOVL    R7,R5                     ; Copy old UCB address
                5C   A5   B7  0A8A  3458         DECW    UCB$W_REFC(R5)            ; Decrement the reference count
          00000000'GF   16  0A8D  3459         JSB     G^IOC$CREDIT_UCB          ; Restore UCB quota to JIB
          00000000'GF   16  0A93  3460         JSB     G^IOC$DELETE_UCB          ; Delete the old UCB
                     28   BA  0A99  3461         POPR    #^M<R3,R5>                ; Restore IRP, UCB address
                50   01   9A  0A9B  3462 70$:    MOVZBL  S^#SS$_NORMAL,R0          ; Return success
                0300 8F   BA  0A9E  3463 80$:    POPR    #^M<R8,R9>                ; Restore registers, R4 is PCB address
                     0B   BB  0AA2  3464         PUSHR   #^M<R0,R1,R3>             ; Save IRP address, status return
          00000000'GF   16  0AA4  3465         JSB     G^SCH$IOUNLOCK            ; Unlock I/O data base
                     0B   BA  0AAA  3466         POPR    #^M<R0,R1,R3>             ; Restore IRP address, status return
                          05  0AAC  3467         RSB                               ; Return to caller
```

```
                        0AAD  3469          .SBTTL  CHECK_PARAM - CHECK SHARED USERS PARAMETERS
                        0AAD  3470 ;++
                        0AAD  3471 ; CHECK_PARAM - CHECK SHARED USERS PARAMETERS
                        0AAD  3472 ;
                        0AAD  3473 ; Functional description:
                        0AAD  3474 ;
                        0AAD  3475 ; Validate all parameters between the requesting SHARED user and the old
                        0AAD  3476 ; existing SHARED user to make sure that are the same.
                        0AAD  3477 ;
                        0AAD  3478 ; Inputs:
                        0AAD  3479 ;       R5 = UCB address of existing shared user
                        0AAD  3480 ;       R7 = UCB address of new shared user
                        0AAD  3481 ;       R8 = Protocol access mode
                        0AAD  3482 ;
                        0AAD  3483 ; Outputs:
                        0AAD  3484 ;       R0 = Status of request
                        0AAD  3485 ;       R1 = Bad parameter code if validation failed
                        0AAD  3486 ;
                        0AAD  3487 ;--
                        0AAD  3488
                        0AAD  3489 CHECK_PARAM:                                 ; Check user parameters
          56    DD      0AAD  3490          PUSHL   R6                          ; Save registers
    50  00D6  C5    9E  0AAF  3491          MOVAB   UCB$B_XE_SHRPRM(R5),R0      ; Get address of current parameters
    56  00D6  C7    9E  0AB4  3492          MOVAB   UCB$B_XE_SHRPRM(R7),R6      ; Get address of new parameters
        52    0B     9A  0AB9  3493          MOVZBL  #UCB$C_XE_SHRPRM,R2         ; Set size of parameter list
                        0ABC  3494 ;
                        0ABC  3495 ; Validate all user settable parameters except for Physical address
                        0ABC  3496 ;
        66    60     91  0ABC  3497 10$:     CMPB    (R0),(R6)                   ; Match?
              2B     12  0ABF  3498          BNEQ    70$                         ; Br if no
        86    80     90  0AC1  3499          MOVB    (R0)+,(R6)+                 ; Store current value in UCB
          F5  52     F5  0AC4  3500          SOBGTR  R2,10$                      ; Loop if more to check
                        0AC7  3501 ;
                        0AC7  3502 ; NOW, check if user has given a hardware physical address.
                        0AC7  3503 ;
                        0AC7  3504 ;       R0 = Address of parameters in UCB
                        0AC7  3505 ;       R6 = Address of parameters in CDB
                        0AC7  3506 ;
                        0AC7  3507          ASSUME  CDB_G_PHA EQ CDB_B_ILP+1
                        0AC7  3508          ASSUME  UCB$G_XE_PHA EQ UCB$B_XE_ILP+1
66      FFFFFFFF  8F    D1  0AC7  3509          CMPL    #-1,(R6)                ; Is user physical address defined?
                08    12  0ACE  3510          BNEQ    20$                     ; Br if yes
 04 A6      FFFF  8F    B1  0AD0  3511          CMPW    #-1,4(R6)               ; Is user physical address defined?
                0F    13  0AD6  3512          BEQL    30$                     ; Br if not
    51      0B04  8F    3C  0AD8  3513 20$:     MOVZWL  #NMA$C_PCLI_PHA,R1      ; Assume bad physical address
            86    80    D1  0ADD  3514          CMPL    (R0)+,(R6)+             ; Physical address match??
                  10    12  0AE0  3515          BNEQ    80$                     ; Br if no
            86    80    B1  0AE2  3516          CMPW    (R0)+,(R6)+             ; Still match??
                  0B    12  0AE5  3517          BNEQ    80$                     ; Br if no
                        0AE7  3518 ;
                        0AE7  3519 ; If this is the shared default user, then set the multicast address list.
                        0AE7  3520 ;
              000F      30  0AE7  3521 30$:     BSBW    SET_MULTIN              ; Set new multicast address list
                09      11  0AEA  3522          BRB     100$                    ; Exit
                        0AEC  3523 ;
                        0AEC  3524 ; Error on parameter validation
                        0AEC  3525 ;
```

```
                51    F6B8 CF42    3C  0AEC  3526 70$:     MOVZWL  BAD_PARAM_TBL-2[R2],R1  ; Return parameter code
                      50   14      9A  0AF2  3527 80$:     MOVZBL  S^#SS$_BADPARAM,R0      ; Return bad parameter error
                           56 8ED0  0AF5  3528 100$:    POPL    R6                      ; Restore registers
                              05    0AF8  3529          RSB
```

```
                                0AF9  3531          .SBTTL  SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
                                0AF9  3532 ;++
                                0AF9  3533 ; SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
                                0AF9  3534 ;
                                0AF9  3535 ; Functional description:
                                0AF9  3536 ;
                                0AF9  3537 ; Copy the multicast address list from the old UCB to the new UCB.
                                0AF9  3538 ; This operation is only done for the SHARED DEFAULT user of the PROTOCOL.
                                0AF9  3539 ;
                                0AF9  3540 ; Inputs:
                                0AF9  3541 ;       R5 = UCB address of existing shared user
                                0AF9  3542 ;       R7 = UCB address of new shared user
                                0AF9  3543 ;       R8 = Protocol access mode
                                0AF9  3544 ;
                                0AF9  3545 ; Outputs:
                                0AF9  3546 ;       R0 = Always success
                                0AF9  3547 ;       All other are preserved.
                                0AF9  3548 ;
                                0AF9  3549 ;--
                                0AF9  3550
                                0AF9  3551 SET_MULTIN:                          ; Set new multicast list
                   58   01   91 0AF9  3552          CMPB    #NMA$C_ACC_SHR,R8   ; Is this the shared default user?
                        13   12 0AFC  3553          BNEQ    50$                 ; Br if not
                        3E   BB 0AFE  3554          PUSHR   #^M<R1,R2,R3,R4,R5> ; Save registers
        00E9 C5 00E9 C7   3C   28 0B00  3555          MOVC3   #6*MAX_C_MLT,UCB$G_XE_MULTI(R7), - ; Copy multicast list
                                0B08  3556                  UCB$G_XE_MULTI(R5)  ;
                        3E   BA 0B08  3557          POPR    #^M<R1,R2,R3,R4,R5> ; Restore registers
                   00E7 C7   90 0B0A  3558          MOVB    UCB$B_XE_MULTI(R7),- ; Copy the number of valid addresses
                   00E7 C5      0B0E  3559                  UCB$B_XE_MULTI(R5)  ;
                   50   01   9A 0B11  3560 50$:     MOVZBL  #SS$_NORMAL,R0      ; Return success
                             05 0B14  3561          RSB                         ; Return to caller
```

```
                     0B15  3563          .SBTTL  SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
                     0B15  3564 ;++
                     0B15  3565 ; SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
                     0B15  3566 ;
                     0B15  3567 ; Functional description:
                     0B15  3568 ;
                     0B15  3569 ; Process read status and read counters requests.
                     0B15  3570 ;
                     0B15  3571 ; The QIO parameters for SENSEMODE are:
                     0B15  3572 ;
                     0B15  3573 ;       P1 = Optional address of quadword buffer
                     0B15  3574 ;       P2 = Optional address of buffer descriptor for extended characteristics
                     0B15  3575 ;
                     0B15  3576 ; The SUBFUNCTION modifiers are as follows:
                     0B15  3577 ;
                     0B15  3578 ; 1)    READ PARAMETERS -- NO MODIFIER.
                     0B15  3579 ;       This function reads the DEUNA/DELUA parameters and returns them to the
                     0B15  3580 ;       user.
                     0B15  3581 ;
                     0B15  3582 ; 2)    READ COUNTERS -- IO$M_RD_COUNT SET.
                     0B15  3583 ;       This function reads the DEUNA/DELUA counters and returns them to the
                     0B15  3584 ;       user.
                     0B15  3585 ;
                     0B15  3586 ;       CLEAR COUNTERS -- IO$M_CLR_COUNT SET.
                     0B15  3587 ;       This modifier must be used with the read counters modifier to clear
                     0B15  3588 ;       the counters as they are read.
                     0B15  3589 ;
                     0B15  3590 ;
                     0B15  3591 ; Inputs:
                     0B15  3592 ;
                     0B15  3593 ;       R3 = IRP address
                     0B15  3594 ;       R4 = PCB address
                     0B15  3595 ;       R5 = UCB address
                     0B15  3596 ;       R6 = CCB address
                     0B15  3597 ;       R7 = Function code
                     0B15  3598 ;       AP = Address of first function-dependent QIO parameter
                     0B15  3599 ;
                     0B15  3600 ;       IPL = ASTDEL
                     0B15  3601 ;
                     0B15  3602 ; Outputs:
                     0B15  3603 ;
                     0B15  3604 ;       R0 = Status return of SENSEMODE request
                     0B15  3605 ;
                     0B15  3606 ;       R1,R2,R6,R7 are destroyed.
                     0B15  3607 ;
                     0B15  3608 ;--
                     0B15  3609
                     0B15  3610 SENSEMODE_FDT::                             ; SENSE MODE I/O FDT processing
50   0084 CF    3C   0B15  3611          MOVZWL  #SS$_DEVOFFLINE,R0          ; Assume unit if offline
           04    E1  0B1A  3612          BBC     #UCB$V_ONLINE,-             ; Br if unit not online
     51 64 A5                0B1C  3613                  UCB$W_STS(R5),15$    ;
  57   20 A3    B0  0B1F  3614          MOVW    IRP$W_FUNC(R3),R7           ; Get entire function code
  14 57    09    E0  0B23  3615          BBS     #IO$V_CTRL,R7,5$            ; Br if line request
                     0B27  3616 ;
                     0B27  3617 ; Check if read circuit counters
                     0B27  3618 ;
  03 57    08    E1  0B27  3619          BBC     #IO$V_RD_COUNT,R7,3$        ; Br if not read circuit counters
```

```
                00DD   31  0B2B  3620          BRW     READ_CIRC_CTR           ; Else, get the circuit counters
           50     01   9A  0B2E  3621 3$:      MOVZBL  S^#SS$_NORMAL,R0        ; Return success
      51   44   A5   D0  0B31  3622           MOVL    UCB$L_DEVDEPEND(R5),R1  ; Get device dependent information
00000000'GF        17  0B35  3623           JMP     G^EXE$FINISHIO          ; Complete the I/O request
                       0B3B  3624
   34 57     08   E1  0B3B  3625 5$:       BBC     #IO$V_RD_COUNT,R7,20$   ; Br if not read counters
                       0B3F  3626 ;
                       0B3F  3627 ; Read counters - modifier RD_COUNT
                       0B3F  3628 ;
                028F   30  0B3F  3629           BSBW    CHECK_P2                ; Check P2 buffer
           50     14   9A  0B42  3630           MOVZBL  S^#SS$_BADPARAM,R0      ; Assume zero length buffer
   38 A3     51   B0  0B45  3631           MOVW    R1,IRP$W_XE_USERSIZ(R3) ; Save user size of P2 buffer
                25   13  0B49  3632           BEQL    15$                     ; Br if none
      51   6A 8F   9A  0B4B  3633           MOVZBL  #LINE_CNT_BUFSIZ-       ; Get size of counter P2 buffer
                       0B4F  3634                   +LINE_CTR_BUFSIZ,R1     ;
                02B9   30  0B4F  3635           BSBW    ALLOC_P2BUF             ; Allocate a buffer
              1B 50    E9  0B52  3636           BLBC    R0,15$                  ; Br if error
      51   2C A3     D0  0B55  3637           MOVL    IRP$L_SVAPTE(R3),R1     ; Get system P2 buffer addr
   04 A1     52   D0  0B59  3638           MOVL    R2,P2B_L_BUFFER(R1)     ; Save user's P2 buffer addr
   3A A3     01   B0  0B5D  3639           MOVW    S^#SS$_NORMAL,IRP$W_XE_STATUS(R3) ; Assume success
   21 A3     0A   90  0B61  3640           MOVB    S^#XE_FC_V_RDCNTS,IRP$B_XE_FUNC(R3) ; Assume Read counts
   04 57     0A   E1  0B65  3641           BBC     #IO$V_CLR_COUNT,R7,10$  ; Br if not clear counts
   21 A3     0B   90  0B69  3642           MOVB    S^#XE_FC_V_RCCNTS,IRP$B_XE_FUNC(R3) ; Else, Read and Clear
                0088   31  0B6D  3643 10$:      BRW     QUEPKT                  ; Queue packet to driver
                F810   31  0B70  3644 15$:      BRW     ABORTIO                 ; Abort the I/O request
                       0B73  3645 ;
                       0B73  3646 ; Read parameters - no modifier
                       0B73  3647 ;
      44 A3     B4   0B73  3648 20$:      CLRW    IRP$W_XE_P2SIZ(R3)      ; No return data
54    0180 C5     D0  0B76  3649           MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
      51   08     3C  0B7B  3650           MOVZWL  S^#8,R1                 ; Size of P1 buffer if present
              024E   30  0B7E  3651           BSBW    CHECK_BUFS              ; Check P1 and P2 buffers
           50   01   9A  0B81  3652           MOVZBL  S^#SS$_NORMAL,R0        ; Assume success
   3A A3     50   B0  0B84  3653           MOVW    R0,IRP$W_XE_STATUS(R3)  ;       ...
   38 A3     51   B0  0B88  3654           MOVW    R1,IRP$W_XE_USERSIZ(R3) ; Save user P2 buffer length
                47   13  0B8C  3655           BEQL    40$                     ; Br if no P2 buffer present
                       0B8E  3656
   40 A3     52   D0  0B8E  3657           MOVL    R2,IRP$L_XE_P2BUF(R3)   ; Save user P2 buffer address
              223F   30  0B92  3658           BSBW    RETURN_P2               ; Return the P2 parameters
   44 A3     50   B0  0B95  3659           MOVW    R0,IRP$W_XE_P2SIZ(R3)   ; Set size of return data
                54   D5  0B99  3660           TSTL    R4                      ; Is CDB present?
                2C   13  0B9B  3661           BEQL    30$                     ; Br if no - okay to return now
      51   50   0A   C1  0B9D  3662           ADDL3   S^#10,R0,R1             ; Check if default physical
                       0BA1  3663                                           ;  address can fit
   38 A3     51   B1  0BA1  3664           CMPW    R1,IRP$W_XE_USERSIZ(R3)         ; Is buffer big enough for
                       0BA5  3665                                           ; default physical address?
                1C   1A  0BA5  3666           BGTRU   25$                     ; Br if no
   44 A3     0A   A0  0BA7  3667           ADDW    #10,IRP$W_XE_P2SIZ(R3)  ; Add DPA to return buffer size
52 40 A3     50   C1  0BAB  3668           ADDL3   R0,IRP$L_XE_P2BUF(R3),R2        ; Get buffer address
                       0BB0  3669                                           ; past end of return data
82 00061488 8F     D0  0BB0  3670           MOVL    #<6@16>+NMA$C_PCLI_HWA!-        ; Store parameter code + size
                       0BB7  3671                   PRM_TYP_M_STRING,(R2)+  ;  ..indicate string parameter
   82   0269 C4     D0  0BB7  3672           MOVL    CDB_G_HWA(R4),(R2)+     ; Store Default Physical Address
   82   026D C4     B0  0BBC  3673           MOVW    CDB_G_HWA+4(R4),(R2)+   ;
                06   11  0BC1  3674           BRB     30$                     ; All is okay
3A A3     0601 8F     B0  0BC3  3675 25$:      MOVW    #SS$_BUFFEROVF,IRP$W_XE_STATUS(R3) ; Return partial success
     50   44 A3     3C  0BC9  3676 30$:      MOVZWL  IRP$W_XE_P2SIZ(R3),R0   ; Get size of user return data
```

```
                50    50    10    78   0BCD  3677          ASHL    #16,R0,R0                ; Shift size of buffer return
                  50    3A A3    B0   0BD1  3678          MOVW    IRP$W_XE_STATUS(R3),R0   ; Get status
                                   0BD5  3679
                              54    D5   0BD5  3680 40$:    TSTL    R4                       ; Is there a CDB?
                              15    13   0BD7  3681          BEQL    50$                      ; Br if no CDB yet!
                  52    3C A3    D0   0BD9  3682          MOVL    IRP$L_XE_USERBUF(R3),R2  ; Retrieve P1 buffer address
                              0F    13   0BDD  3683          BEQL    50$                      ; Br if none
                  62    40 A5    7D   0BDF  3684          MOVQ    UCB$B_DEVCLASS(R5),(R2)  ; Else, return characteristics
            04 A2    00C4 C4    C8   0BE3  3685          BISL    CDB_L_DEVDEPEND(R4),4(R2) ;     ...
                51    00C4 C4    D0   0BE9  3686          MOVL    CDB_L_DEVDEPEND(R4),R1   ; Get device dependent info
                  51    44 A5    C8   0BEE  3687 50$:    BISL    UCB$L_DEVDEPEND(R5),R1   ; ..from UCB also
                 00000000'GF    17   0BF2  3688          JMP     G^EXE$FINISHIO           ; Complete the I/O request
                                   0BF8  3689
                                   0BF8  3690 ;
                                   0BF8  3691 ; Queue I/O request to driver
                                   0BF8  3692 ;
                                   0BF8  3693 QUEPKT:                                   ; Queue packet to driver
                                   0BF8  3694          SETIPL  UCB$B_FIPL(R5)           ; Raise IPL to fork IPL
                 00000000'GF    16   0BFC  3695          JSB     G^IOC$INITIATE           ; Intiate the I/O request
                 00000000'GF    17   0C02  3696          JMP     G^EXE$QIORETURN          ; Lower IPL, and REI
```

```
                                                0C08  3698          .SBTTL  READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
                                                0C08  3699 ;++
                                                0C08  3700 ; READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
                                                0C08  3701 ;
                                                0C08  3702 ; Functional description:
                                                0C08  3703 ;
                                                0C08  3704 ; Process read circuit counters request.
                                                0C08  3705 ;
                                                0C08  3706 ; The QIO parameters for SENSEMODE are:
                                                0C08  3707 ;
                                                0C08  3708 ;       P2 = Address of buffer descriptor for counters
                                                0C08  3709 ;
                                                0C08  3710 ;
                                                0C08  3711 ; Inputs:
                                                0C08  3712 ;
                                                0C08  3713 ;       R3 = IRP address
                                                0C08  3714 ;       R4 = PCB address
                                                0C08  3715 ;       R5 = UCB address
                                                0C08  3716 ;       R6 = CCB address
                                                0C08  3717 ;       R7 = Function code and modifier bits
                                                0C08  3718 ;       AP = Address of first function-dependent QIO parameter
                                                0C08  3719 ;
                                                0C08  3720 ; Outputs:
                                                0C08  3721 ;
                                                0C08  3722 ;       R0 = Status return of SENSEMODE request
                                                0C08  3723 ;
                                                0C08  3724 ;       R1,R2,R6,R7 are destroyed.
                                                0C08  3725 ;
                                                0C08  3726 ;--
                                                0C08  3727
                                                0C08  3728 ABORT_IRP:
                                     F778   31  0C08  3729          BRW     ABORTIO                 ; Abort the I/O request
                                                0C0B  3730
                                                0C0B  3731 READ_CIRC_CTR:                          ; Read the circuit counters
                                          01C3  30  0C0B  3732          BSBW    CHECK_P2                ; Check the P2 buffer
                                       50   14  9A  0C0E  3733          MOVZBL  S^#SS$_BADPARAM,R0      ; Assume zero length buffer
                                    38 A3   51  B0  0C11  3734          MOVW    R1,IRP$W_XE_USERSIZ(R3) ; Save size of user P2 buffer
                                           F1  13  0C15  3735          BEQL    ABORT_IRP               ; Br if no buffer
                                       51   20  9A  0C17  3736          MOVZBL  #CIRC_CTR_BUFSIZ,R1     ; Get size of system P2 buffer
                                          01EE  30  0C1A  3737          BSBW    ALLOC_P2BUF             ; Allocate the buffer
                                        E8 50   E9  0C1D  3738          BLBC    R0,ABORT_IRP            ; Br if error
                                    51   2C A3   D0  0C20  3739          MOVL    IRP$L_SVAPTE(R3),R1     ; Get system P2 buffer address
                                    04 A1   52  D0  0C24  3740          MOVL    R2,P2B_L_BUFFER(R1)     ; Save user P2 buffer address
                                       52   61  D0  0C28  3741          MOVL    P2B_L_POINTER(R1),R2    ; Get address of data portion of buffer
                                                0C2B  3742          PUSHQ   R8                      ; Save R8, R9
                                                0C2E  3743 ;
                                                0C2E  3744 ; Get the counters kept by the driver
                                                0C2E  3745 ;
                                 58     F547 CF  9E  0C2E  3746          MOVAB   CIRC_CTR,R8             ; Get address of counter format table
                                       51   06  9A  0C33  3747          MOVZBL  #CIRC_CTR_SIZE,R1       ; Get number of entries in table
                                       50   88  3C  0C36  3748 20$:     MOVZWL  (R8)+,R0                ; Get counter code
                                       82   50  B0  0C39  3749          MOVW    R0,(R2)+                ; Return counter type code
                                       59   88  3C  0C3C  3750          MOVZWL  (R8)+,R9                ; Get offset word
                                       59   55  C0  0C3F  3751          ADDL    R5,R9                   ; Point to counter in UCB
                                 50    50   03  0C  EF  0C42  3752          EXTZV   #NMA$V_CNT_MAP,#3,R0,R0 ; Get width + bit map indicator
                                                0C47  3753          CASE    R0,TYPE=B,LIMIT=#2,<-   ; Dispatch on width and bit map
                                                0C47  3754                  30$,-                   ; 8 bit counter
```

```
                     0C47  3755                 30$,-                   ; 8 bit counter + bit map
                     0C47  3756                 40$,-                   ; 16 bit counter
                     0C47  3757                 30$,-                   ; 16 bit counter + bit map
                     0C47  3758                 35$>                    ; 32 bit counter
                     0C55  3759
                     0C55  3760 30$:    BUG_CHECK NOBUFPCKT,FATAL       ; All others, fatal error
                     0C59  3761 ;
                     0C59  3762 ; 32 BIT counter
                     0C59  3763 ;
   82    89    B0    0C59  3764 35$:    MOVW    (R9)+,(R2)+             ; Store counter in buffer
03 57    0A    E1    0C5C  3765         BBC     #IO$V_CLR_COUNT,R7,40$  ; Br if not clear counter operation
     FE A9    B4    0C60  3766         CLRW    -2(R9)                  ; Else, clear the counter as well
                     0C63  3767 ;
                     0C63  3768 ; 16 BIT counter
                     0C63  3769 ;
   82    69    B0    0C63  3770 40$:    MOVW    (R9),(R2)+              ; Store counter in buffer
02 57    0A    E1    0C66  3771         BBC     #IO$V_CLR_COUNT,R7,50$  ; Br if not clear counter operation
         69    B4    0C6A  3772         CLRW    (R9)                    ; Else, clear the counter as well
      C7 51    F5    0C6C  3773 50$:    SOBGTR  R1,20$                  ; Loop if more
                     0C6F  3774         POPQ    R8                      ; Restore R8, R9
   51    01    9A    0C72  3775         MOVZBL  S^#SS$_NORMAL,R1        ; Assume success
     38 A3    B1    0C75  3776         CMPW    IRP$W_XE_USERSIZ(R3),-  ; Is user's buffer big enough?
     32 A3          0C78  3777                 IRP$W_BCNT(R3)          ;
         0A    1E    0C7A  3778         BGEQU   60$                     ; Br if yes
     38 A3    B0    0C7C  3779         MOVW    IRP$W_XE_USERSIZ(R3),-  ; Else, set size to minimum
     32 A3          0C7F  3780                 IRP$W_BCNT(R3)          ; Of both
51   0601 8F    B0    0C81  3781         MOVW    #SS$_BUFFEROVF,R1       ; Set partial success
50     30 A3    D0    0C86  3782 60$:    MOVL    IRP$Q_BCNT-2(R3),R0     ; Get size of buffer returned in
                     0C8A  3783                                         ; ..high word of R0
   50    51    B0    0C8A  3784         MOVW    R1,R0                   ; Get status return
51     44 A5    D0    0C8D  3785         MOVL    UCB$L_DEVDEPEND(R5),R1  ; Get device dependent info
00000000'GF    17    0C91  3786         JMP     G^EXE$FINISHIO          ; Complete the I/O request
```

```
                                                K 4
0C97  3788          .SBTTL  DIAGNOSE_FDT - DIAGNOSTIC OPERATION FDT DISPATCH ROUTINE
0C97  3789 ;++
0C97  3790 ; DIAGNOSE_FDT - DIAGNOSTIC FDT PROCESSING
0C97  3791 ;
0C97  3792 ; Functional description:
0C97  3793 ;
0C97  3794 ; This is the fdt routine for diagnostic functions.
0C97  3795 ; There are several functions based on function code passed in P3 of the
0C97  3796 ; parameter list.
0C97  3797 ;
0C97  3798 ; The QIO parameters for DIAGNODE are:
0C97  3799 ;
0C97  3800 ;       P1 = Optional address of a diagnostic buffer
0C97  3801 ;       P2 = Optional length of the diagnostic bu'fer
0C97  3802 ;       P3 = Diagnost function code
0C97  3803 ;       R4 = Optional address of DEUNA/DELUA micro-processor internal address
0C97  3804 ;
0C97  3805 ;
0C97  3806 ; Inputs:
0C97  3807 ;
0C97  3808 ;       R3 = IRP ADDRESS
0C97  3809 ;       R4 = PCB ADDRESS
0C97  3810 ;       R5 = UCB ADDRESS
0C97  3811 ;       R6 = CCB ADDRESS
0C97  3812 ;       R7 = FUNCTION CODE
0C97  3813 ;       AP = ADDRESS OF THE FIRST QIO PARAMETER
0C97  3814 ;
0C97  3815 ;       IPL = ASTDEL
0C97  3816 ;
0C97  3817 ; Outputs:
0C97  3818 ;
0C97  3819 ;       R3 = IRP ADDRESS
0C97  3820 ;       R4 = PCB ADDRESS
0C97  3821 ;       R5 = UCB ADDRESS
0C97  3822 ;
0C97  3823 ;       R0-R2,R6 are destroyed.
0C97  3824 ;
0C97  3825 ;--
0C97  3826
0C97  3827 DIAGNOSE_FDT::                                  ; DIAGNOSE FDT processing
      2C A3    D4  0C97  3828          CLRL    IRP$L_SVAPTE(R3)        ; Set no buffered packet
      38 A3    7C  0C9A  3829          CLRQ    IRP$L_MEDIA(R3)         ; Reset mode data area
50    0084 8F    3C  0C9D  3830          MOVZWL  #SS$_DEVOFFLINE,R0      ; Assume unit is offline
         04    E1  0CA2  3831          BBC     #UCB$V_ONLINE,-         ; Br if unit not online
   23 64 A5          0CA4  3832                  UCB$W_STS(R5),DG_ERR    ;
3A A3    01    9B  0CA7  3833          MOVZBW  S^#SS$_NORMAL,IRP$W_XE_STATUS(R3) ; Assume success
51    08 AC    9A  0CAB  3834          MOVZBL  P3(AP),R1               ; Get Diagnostic function
                   0CAF  3835          $DISPATCH       R1,TYPE=B,-     ; Distpatch on function code
                   0CAF  3836          <-      ;function       action
                   0CAF  3837
                   0CAF  3838                  <XE$C_DG_LOADMEM LOADMEM>,- ; Load DEUNA/DELUA memory
                   0CAF  3839                  <XE$C_DG_READMEM READMEM>,- ; Read DEUNA/DELUA memory
                   0CAF  3840                  <XE$C_DG_UPADDR  UPADDR>,-  ; Load and start micro-processor
                   0CAF  3841                  <XE$C_DG_WRITEID WRITEID>,- ; Write SYSTEM ID parameters
                   0CAF  3842                  <XE$C_DG_READID  READID>,-  ; Read SYSTEM ID parameters
                   0CAF  3843                  <XE$C_DG_STOP    STOPPR>,-  ; Stop xmit and receive process
                   0CAF  3844                  <XE$C_DG_STEST   STEST>,-   ; Perfore a self test
```

```
                     OCAF  3845                 <XE$C_DG_READST  READST>,-  ; Read port status
                     OCAF  3846                 <XE$C_DG_ESTEST  ESTEST>,-  ; Perform a DELUA self test
                     OCAF  3847                 <XE$C_DG_HALT    HALT>,-    ; Halt port (DELUA only)
                     OCAF  3848             >
                     OCC7  3849 ;
                     OCC7  3850 ; All other functions
                     OCC7  3851 ;
                     OCC7  3852 DG_ERR1:
          50   14   9A  OCC7  3853             MOVZBL  #SS$_BADPARAM,R0        ; Bad parameter
             F6B6   31  OCCA  3854 DG_ERR:     BRW     ABORTIO                 ; Abort the I/O
                     OCCD  3855 ;
                     OCCD  3856 ; Load DEUNA/DELUA memory
                     OCCD  3857 ;
                     OCCD  3858 ;       P1 = Address of buffer to write micro-code into DEUNA/DELUA.
                     OCCD  3859 ;       P2 = Length of P1 buffer
                     OCCD  3860 ;       P4 = Internal DEUNA/DELUA address to start loading at
                     OCCD  3861 LOADMEM:
             019D   30  OCCD  3862             BSBW    CHECK_DGBUFR            ; Check buffer for read access
 44 A3   0C AC   B0  OCD0  3863             MOVW    P4(AP),IRP$L_XE_UPADR(R3) ; Save micro-processor address
    EE 44 A3   E8  OCD5  3864             BLBS    IRP$L_XE_UPADR(R3),DG_ERR1 ; Br if bad address
 21 A3   0D   90  OCD9  3865             MOVB    #XE_FC_V_LDMEM,IRP$B_XE_FUNC(R3) ; Set function request
             FF18   31  OCDD  3866             BRW     QUEPKT                  ; Queue request to driver
                     OCE0  3867 ;
                     OCE0  3868 ; Read DEUNA/DELUA memory
                     OCE0  3869 ;
                     OCE0  3870 ;       P1 = Address of buffer to read DEUNA/DELUA memory into
                     OCE0  3871 ;       P2 = Length of P1 buffer
                     OCE0  3872 ;       P4 = Internal DEUNA/DELUA address to start reading at
                     OCE0  3873 READMEM:
             0192   30  OCE0  3874             BSBW    CHECK_DGBUFW            ; Check buffer for write access
 44 A3   0C AC   B0  OCE3  3875             MOVW    P4(AP),IRP$L_XE_UPADR(R3) ; Save micro-processor address
    DB 44 A3   E8  OCE8  3876             BLBS    IRP$L_XE_UPADR(R3),DG_ERR1 ; Br if bad address
 21 A3   0E   90  OCEC  3877             MOVB    #XE_FC_V_RDMEM,IRP$B_XE_FUNC(R3) ; Set function request
             FF05   31  OCF0  3878             BRW     QUEPKT                  ; Queue request to driver
                     OCF3  3879 ;
                     OCF3  3880 ; Load and start at micro-processor address
                     OCF3  3881 ;
                     OCF3  3882 ;       P1 = unused
                     OCF3  3883 ;       P2 = unused
                     OCF3  3884 ;       P4 = Internal DEUNA/DELUA address to start at
                     OCF3  3885 UPADDR:
 44 A3   0C AC   B0  OCF3  3886             MOVW    P4(AP),IRP$L_XE_UPADR(R3) ; Save micro-processor address
    CB 44 A3   E8  OCF8  3887             BLBS    IRP$L_XE_UPADR(R3),DG_ERR1 ; Br if bad address
 21 A3   0F   90  OCFC  3888             MOVB    #XE_FC_V_UPST,IRP$B_XE_FUNC(R3) ; Set function request
             FEF5   31  OD00  3889             BRW     QUEPKT                  ; Queue request to driver
                     OD03  3890 ;
                     OD03  3891 ; Write SYSTEM ID parameters
                     OD03  3892 ;
                     OD03  3893 ;       P1 = Address of buffer containing new SYSTEM ID parameters
                     OD03  3894 ;       P2 = Length of P1 buffer
                     OD03  3895 ;       P4 = unused
                     OD03  3896 WRITEID:
             0167   30  OD03  3897             BSBW    CHECK_DGBUFR            ; Check buffer for read access
          50   14   9A  OD06  3898             MOVZBL  #SS$_BADPARAM,R0        ; Assume bad parameter
 32 A3   08   B1  OD09  3899             CMPW    #8,IRP$W_BCNT(R3)       ; Is buffer size good?
             03   1B  OD0D  3900             BLEQU   10$                     ; Br if yes
             F671   31  OD0F  3901             BRW     ABORTIO                 ; Else, abort the I/O request
```

```
                     0D12  3902 10$:    MOVB    #XE_FC_V_LDSID,IRP$B_XE_FUNC(R3) ; Set function request
FEDF   31            0D16  3903         BRW     QUEPKT                           ; Queue request to driver
                     0D19  3904 ;
                     0D19  3905 ; Read SYSTEM ID parameters
                     0D19  3906 ;
                     0D19  3907 ;       P1 = Address of buffer for reading SYSTEM ID parameters
                     0D19  3908 ;       P2 = Length of P1 buffer
                     0D19  3909 ;       P4 = unused
                     0D19  3910 READID:
0159   30            0D19  3911         BSBW    CHECK_DGBUFW                     ; Check buffer for write access
21 A3   11   90      0D1C  3912         MOVB    #XE_FC_V_RDSID,IRP$B_XE_FUNC(R3) ; Set function request
FED5   31            0D20  3913         BRW     QUEPKT                           ; Queue request to driver
                     0D23  3914 ;
                     0D23  3915 ; Stop transmit and receive process
                     0D23  3916 ;
                     0D23  3917 ;       P1 = unused
                     0D23  3918 ;       P2 = unused
                     0D23  3919 ;       P4 = unused
                     0D23  3920 STOPPR:
21 A3   12   90      0D23  3921         MOVB    #XE_FC_V_STOPPR,IRP$B_XE_FUNC(R3) ; Set function request
FECE   31            0D27  3922         BRW     QUEPKT                           ; Queue request to driver
                     0D2A  3923 ;
                     0D2A  3924 ; Perform a DEUNA/DELUA self test
                     0D2A  3925 ;
                     0D2A  3926 ;       P1 = unused
                     0D2A  3927 ;       P2 = unused
                     0D2A  3928 ;       P4 = unused
                     0D2A  3929 STEST:
21 A3   13   90      0D2A  3930         MOVB    #XE_FC_V_STEST,IRP$B_XE_FUNC(R3) ; Set function request
FEC7   31            0D2E  3931         BRW     QUEPKT                           ; Queue request to driver
                     0D31  3932 ;
                     0D31  3933 ; Read port status
                     0D31  3934 ;
                     0D31  3935 ;       P1 = Address of buffer for reading the port status
                     0D31  3936 ;       P2 = unused
                     0D31  3937 ;       P4 = unused
                     0D31  3938 READST:
51   08   9A           0D31  3939         MOVZBL  #8,R1                            ; Set size of buffer
0129   30            0D34  3940         BSBW    CHECK_DGBUFA                     ; Check buffer for write access
21 A3   14   90      0D37  3941         MOVB    #XE_FC_V_READST,IRP$B_XE_FUNC(R3) ; Set function request
FEBA   31            0D3B  3942         BRW     QUEPKT                           ; Queue request to driver
                     0D3E  3943 ;
                     0D3E  3944 ; Perform a DELUA extended self test
                     0D3E  3945 ;
                     0D3E  3946 ;       P1 = unused
                     0D3E  3947 ;       P2 = unused
                     0D3E  3948 ;       P4 = unused
                     0D3E  3949 ;
                     0D3E  3950 ESTEST:
51   0A A5   9A      0D3E  3951         MOVZBL  UCB$B_TYPE(R5),R1
                     0D42  3952         $DISPATCH       R1,TYPE=B,-      ; Dispatch based upon device type
                     0D42  3953         < -
                     0D42  3954                 <DT$_DEUNA      10$>,-   ; DEUNA - error
                     0D42  3955                 <DT$_DELUA      20$>,-   ; DELUA - ok to do the test
                     0D42  3956         >
FF66   31            0D5E  3957 10$:    BRW     DG_ERR1                          ; Go return bad parameter error
                     0D61  3958
```

```
                                                N 4
21 A3   1A   90  0D61  3959 20$:    MOVB    #XE_FC_V_ESTEST,IRP$B_XE_FUNC(R3) ; Set function request
      FE90   31  0D65  3960          BRW     QUEPKT                            ; Queue request to driver
                 0D68  3961
                 0D68  3962 ;
                 0D68  3963 ; Halt the DELUA Port
                 0D68  3964 ;
                 0D68  3965 ;       P1 = unused
                 0D68  3966 ;       P2 = unused
                 0D68  3967 ;       P4 = unused
                 0D68  3968 ;
                 0D68  3969 HALT:
51   0A A5   90  0D68  3970          MOVB    UCB$B_TYPE(R5),R1
                 0D6C  3971          $DISPATCH       R1,TYPE=B,-       ; Dispatch based upon device type
                 0D6C  3972          < -
                 0D6C  3973                  <DT$_DEUNA      10$>,-    ; DEUNA - error
                 0D6C  3974                  <DT$_DELUA      20$>,-    ; DELUA - ok to do the test
                 0D6C  3975          >
      FF3C   31  0D88  3976 10$:    BRW     DG_ERR1                           ; Go return bad parameter error
                 0D8B  3977
21 A3   1B   9A  0D8B  3978 20$:    MOVZBL  #XE_FC_V_HALT,IRP$B_XE_FUNC(R3) ; Set function request
      FE66   31  0D8F  3979          BRW     QUEPKT                            ; Queue request to driver
```

```
                                0D92  3981          .SBTTL  GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
                                0D92  3982 ;++
                                0D92  3983 ; GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
                                0D92  3984 ;
                                0D92  3985 ; Functional description:
                                0D92  3986 ;
                                0D92  3987 ; This routine saves the P2 buffer for later use by the driver.
                                0D92  3988 ; The P2 buffer is saved by allocating the appropriate amount of memory from
                                0D92  3989 ; non-paged pool.   The user's quota is checked before the allocation is made.
                                0D92  3990 ; And the non-paged pool buffer is charged against the user's quota. The P2
                                0D92  3991 ; system buffer address is passed in IRP$L_SVAPTE of the IRP.
                                0D92  3992 ;
                                0D92  3993 ;
                                0D92  3994 ; Inputs:
                                0D92  3995 ;
                                0D92  3996 ;       R3 = IRP address
                                0D92  3997 ;       R4 = PCB address
                                0D92  3998 ;       R5 = UCB address
                                0D92  3999 ;
                                0D92  4000 ; Outputs:
                                0D92  4001 ;
                                0D92  4002 ;       R0 = status of buffers
                                0D92  4003 ;
                                0D92  4004 ;       R3-R5 are preserved.
                                0D92  4005 ;
                                0D92  4006 ;--
                                0D92  4007
                                0D92  4008 GET_CHAR_BUF:                                ; Get characteristics buffer
                                0D92  4009 ;
                                0D92  4010 ; Check access to P2 buffer and check process's buffer quota
                                0D92  4011 ;
      51    04 AC    D0         0D92  4012 10$:    MOVL    P2(AP),R1                   ; Get address P2 char buf desc
                33    13        0D96  4013         BEQL    40$                         ; Br if no P2 buffer
                53    DD        0D98  4014         PUSHL   R3                          ; Save R3
      00000000'GF     16        0D9A  4015         JSB     G^EXE$PROBER_DSC            ; Check access to buffer
           CE 50      E9        0DA0  4016         BLBC    R0,15$                      ; Br if error
         51   51     3C         0DA3  4017         MOVZWL  R1,R1                       ; Convert length to longword
                52    DD        0DA6  4018         PUSHL   R2                          ; Save R2
      00000000'GF     16        0DA8  4019         JSB     G^EXE$BUFQUOPRC             ; Check for buffered quota
                52 8ED0         0DAE  4020         POPL    R2                          ; Restore R2
                53 8ED0         0DB1  4021 15$:    POPL    R3                          ; Restore R3
           01 50      E8        0DB4  4022         BLBS    R0,30$                      ; Branch if quota ok
                      05        0DB7  4023 20$:    RSB                                 ; Return
                                0DB8  4024
                                0DB8  4025 ;
                                0DB8  4026 ; Quota OKAY, allocate buffer and copy info.
                                0DB8  4027 ;
              0050      30      0DB8  4028 30$:    BSBW    ALLOC_P2BUF                 ; Allocate buffer
           10 50        E9      0DBB  4029         BLBC    R0,50$                      ; Br if error
      50   2C A3        D0      0DBE  4030         MOVL    IRP$L_SVAPTE(R3),R0         ; Get P2 buffer address
                38      BB      0DC2  4031         PUSHR   #^M<R3,R4,R5>               ; Save sacred registers
   0C A0    62   51      28      0DC4  4032         MOVC3   R1,(R2),P2B_T_DATA(R0)      ; Save P2 char buffer
                38      BA      0DC9  4033         POPR    #^M<R3,R4,R5>               ; Restore registers
           50   01      9A      0DCB  4034 40$:    MOVZBL  S^#SS$_NORMAL,R0            ; Set success
                      05        0DCE  4035 50$:    RSB                                 ; Return
```

```
                     ODCF  4037          .SBTTL  CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
                     ODCF  4038 ;++
                     ODCF  4039 ; CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
                     ODCF  4040 ;
                     ODCF  4041 ; Functional description:
                     ODCF  4042 ;
                     ODCF  4043 ; This routines checks the P1 and P2 buffers for write access if supplied.
                     ODCF  4044 ;
                     ODCF  4045 ; Inputs:
                     ODCF  4046 ;
                     ODCF  4047 ;       R1 = Size of P1 buffer needed for write access
                     ODCF  4048 ;       R3 = IRP address
                     ODCF  4049 ;       R4 = PCB address
                     ODCF  4050 ;       R5 = UCB address
                     ODCF  4051 ;       R7 = Function code
                     ODCF  4052 ;
                     ODCF  4053 ; Outputs:
                     ODCF  4054 ;
                     ODCF  4055 ;       R0 is destroyed.
                     ODCF  4056 ;       R1 = Length of P2 buffer (zero if no P2 buffer)
                     ODCF  4057 ;       R2 = Address of P2 buffer in user's process space
                     ODCF  4058 ;
                     ODCF  4059 ;
                     ODCF  4060 ;       No RETURN on NO ACCESS
                     ODCF  4061 ;
                     ODCF  4062 ; Implicit Outputs:
                     ODCF  4063 ;
                     ODCF  4064 ;       IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
                     ODCF  4065 ;
                     ODCF  4066 ;--
                     ODCF  4067
                     ODCF  4068 CHECK_BUFS:
          27   10    ODCF  4069          BSBB    CHECK_P1                        ; Check P1 buffer
                     ODD1  4070 CHECK_P2:
          51   D4    ODD1  4071          CLRL    R1                              ; Assume no P2 buffer desc
    52    04 AC   D0  ODD3  4072          MOVL    P2(AP),R2                       ; Get address of P2 desc
          18   13    ODD7  4073          BEQL    10$                             ; Br if no P2
                     ODD9  4074          IFNORD  #8,(R2),ACCESS                  ; Br if no access
    51    62   3C    ODDF  4075          MOVZWL  (R2),R1                         ; Get length of buffer
          0D   13    ODE2  4076          BEQL    10$                             ; Br if zero
 50    04 A2   D0    ODE4  4077          MOVL    DSC$A_POINTER(R2),R0            ; Get buffer address
00000000'GF   16    ODE8  4078          JSB     G^EXE$READCHK                   ; Check write access to buffer
                     ODEE  4079                                                  ; (no return no access)
                     ODEE  4080                                                  ; Also sets IRP$V_FUNC in IRP
    52   50   D0    ODEE  4081          MOVL    R0,R2                           ; Copy buffer address
         05   05    ODF1  4082 10$:     RSB                                     ; Return to caller
                     ODF2  4083
    50   0C   9A    ODF2  4084 ACCESS:  MOVZBL  S^#SS$_ACCVIO,R0                ; Return access violation
       F588   31    ODF5  4085          BRW     ABORTIO                         ; Abort the I/O request
```

```
                        0DF8  4087          .SBTTL  CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
                        0DF8  4088 ;++
                        0DF8  4089 ; CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
                        0DF8  4090 ;
                        0DF8  4091 ; Functional description:
                        0DF8  4092 ;
                        0DF8  4093 ; This routine checks the P1 buffer and if okay, the buffer address
                        0DF8  4094 ; is saved in IRP$L_MEDIA of the IRP.
                        0DF8  4095 ;
                        0DF8  4096 ; Inputs:
                        0DF8  4097 ;
                        0DF8  4098 ;       R1 = Size of buffer for write access
                        0DF8  4099 ;       R3 = IRP address
                        0DF8  4100 ;       R4 = PCB address
                        0DF8  4101 ;       R5 = UCB address
                        0DF8  4102 ;       R7 = Function code
                        0DF8  4103 ;
                        0DF8  4104 ; Outputs:
                        0DF8  4105 ;
                        0DF8  4106 ;       R0 is destroyed.
                        0DF8  4107 ;
                        0DF8  4108 ;       No RETURN on NO ACCESS.
                        0DF8  4109 ;
                        0DF8  4110 ; Implicit Outputs:
                        0DF8  4111 ;
                        0DF8  4112 ;       IRP$L_MEDIA(R3) = User P1 buffer address.
                        0DF8  4113 ;       IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
                        0DF8  4114 ;
                        0DF8  4115 ;--
                        0DF8  4116
                        0DF8  4117 CHECK_P1:
      38 A3     D4      0DF8  4118          CLRL    IRP$L_MEDIA(R3)                 ; Assume no P1 buffer
   50    6C     D0      0DFB  4119          MOVL    P1(AP),R0                       ; Get address of user buffer
         06     13      0DFE  4120          BEQL    10$                             ; Br if none
00000000'GF     16      0E00  4121          JSB     G^EXE$READCHK                   ; Check access to buffer
                        0E06  4122                                                  ; (No return - no access)
 3C A3    50     D0      0E06  4123 10$:     MOVL    R0,IRP$L_XE_USERBUF(R3)         ; Save P1 buffer address in IRP
                 05      0E0A  4124          RSB                                     ; Return to caller
```

```
                        0E0B  4126          .SBTTL  ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
                        0E0B  4127 ;++
                        0E0B  4128 ; ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
                        0E0B  4129 ;
                        0E0B  4130 ; Functional description:
                        0E0B  4131 ;
                        0E0B  4132 ; This routine allocates a system buffer and returns the address in the IRP at
                        0E0B  4133 ; IRP$L_SVAPTE.  The size of the allocation, including buffer header must
                        0E0B  4134 ; be at least 24 bytes in length.
                        0E0B  4135 ;
                        0E0B  4136 ; Inputs:
                        0E0B  4137 ;
                        0E0B  4138 ;       R1 = Size of allocation desired
                        0E0B  4139 ;       R3 = IRP address
                        0E0B  4140 ;
                        0E0B  4141 ; Outputs:
                        0E0B  4142 ;
                        0E0B  4143 ;       R0 = status of request
                        0E0B  4144 ;
                        0E0B  4145 ;       R1-R5 are preserved.
                        0E0B  4146 ;
                        0E0B  4147 ; Implicit Outputs:
                        0E0B  4148 ;
                        0E0B  4149 ;       IRP$L_SVAPTE(R3) = address of system buffer
                        0E0B  4150 ;       IRP$W_BOFF(R3) = byte count charged to user's process
                        0E0B  4151 ;       IRP$W_BCNT(R3) = original byte count requested
                        0E0B  4152 ;
                        0E0B  4153 ;       All parts of the P2 buffer header are initialized, except for the
                        0E0B  4154 ;       user's P2 buffer address.
                        0E0B  4155 ;
                        0E0B  4156 ;--
                        0E0B  4157
                        0E0B  4158 ALLOC_P2BUF:                                 ; Allocate a non-paged buffer
        51    D5        0E0B  4159          TSTL    R1                          ; Zero length buffer?
        4D    13        0E0D  4160          BEQL    30$                         ; Br if yes
        0E    BB        0E0F  4161          PUSHR   #^M<R1,R2,R3>               ; Save registers
 32 A3  51    B0        0E11  4162          MOVW    R1,IRP$W_BCNT(R3)           ; Save original byte count
    0C  51    D1        0E15  4163          CMPL    R1,S^#24=P2B_C_LENGTH       ; Is buffer big enough?
        03    1A        0E18  4164          BGTRU   5$                          ; Br if yes
    51  0C    D0        0E1A  4165          MOVL    S^#24-P2B_C_LENGTH,R1       ; Else, set size to minimum
    51  0C    C0        0E1D  4166 5$:      ADDL2   S^#P2B_C_LENGTH,R1          ; Add in size of header
00000000'GF   16        0E20  4167          JSB     G^EXE$BUFQUOPRC             ; Check for buffered quota
      0E 50   E9        0E26  4168          BLBC    R0,10$                      ; Branch if quota bad
                        0E29  4169 ;
                        0E29  4170 ; Quota OKAY, allocate buffer and copy info.
                        0E29  4171 ;
        51    DD        0E29  4172          PUSHL   R1                          ; Save size to charge user
00000000'GF   16        0E2B  4173          JSB     G^EXE$ALLOCBUF              ; Go allocate a buffer
      06 50   E8        0E31  4174          BLBS    R0,20$                      ; Br if success
    5E    04    C0      0E34  4175          ADDL    #4,SP                       ; Pop saved size
        0E    BA        0E37  4176 10$:     POPR    #^M<R1,R2,R3>               ; Restore registers
              05        0E39  4177          RSB                                 ; Return with error code in R0
                        0E3A  4178 ;
                        0E3A  4179 ; System buffer allocated decrement user's quota
                        0E3A  4180 ;
        53 8ED0         0E3A  4181 20$:     POPL    R3                          ; Restore user quota charge
 62   0C A2    9E        0E3D  4182          MOVAB   P2B_T_DATA(R2),P2B_L_POINTER(R2) ; Set address to start of data
```

```
                08 A2    53   B0  0E41  4183          MOVW    R3,P2B_W_SIZE(R2)               ; Save buffer size in buffer
                   50    52   D0  0E45  4184          MOVL    R2,R0                           ; Save P2 char buf addr
             52    0080  C4   D0  0E48  4185          MOVL    PCB$L_JIB(R4),R2                ; Get JIB address
                20 A2    53   C2  0E4D  4186          SUBL    R3,JIB$L_BYTCNT(R2)             ; Decrement user's quota
                         0E   BA  0E51  4187          POPR    #^M<R1,R2,R3>                   ; Restore registers
                2C A3    50   D0  0E53  4188          MOVL    R0,IRP$L_SVAPTE(R3)             ; Save P2 buffer address in IRP
             30 A3    08 A0   B0  0E57  4189          MOVW    P2B_W_SIZE(R0),IRP$W_BOFF(R3)   ; Return buffer size in IRP
                   50    01   9A  0E5C  4190 30$:     MOVZBL  S^#SS$_NORMAL,R0                ; Set success
                               05  0E5F  4191          RSB                                     ; Return to caller
```

```
                        0E60  4193          .SBTTL  CHECK_DGBUFx - CHECK ACCESS TO DIAGNOSTIC BUFFER
                        0E60  4194 ;++
                        0E60  4195 ; CHECK_DGBUFx - CHECK ACCESS TO DIAGNOSTIC BUFFER
                        0E60  4196 ;
                        0E60  4197 ; Functional description:
                        0E60  4198 ;
                        0E60  4199 ; This routines checks access to a diagnostic buffer as part of a diagnostic
                        0E60  4200 ; function request.
                        0E60  4201 ;
                        0E60  4202 ; Inputs:
                        0E60  4203 ;
                        0E60  4204 ;       R1 = Length of buffer (CHECK_DGBUFA only)
                        0E60  4205 ;       R3 = IRP address
                        0E60  4206 ;       R4 = PCB address
                        0E60  4207 ;       R5 = UCB address
                        0E60  4208 ;
                        0E60  4209 ; Outputs:
                        0E60  4210 ;
                        0E60  4211 ;       R0,R1,R2 are destroyed.
                        0E60  4212 ;
                        0E60  4213 ;
                        0E60  4214 ;       No RETURN on NO ACCESS
                        0E60  4215 ;
                        0E60  4216 ; Implicit Outputs:
                        0E60  4217 ;
                        0E60  4218 ;       If CHECK_DGBUFW, then IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK
                        0E60  4219 ;       subroutine.
                        0E60  4220 ;
                        0E60  4221 ;--
                        0E60  4222          .ENABL  LSB
                        0E60  4223 CHECK_DGBUFA:                            ; Check write buffer to diag buffer
                        0E60  4224                                          ;  - Address check only
     50    6C    D0     0E60  4225          MOVL    P1(AP),R0               ; Get address of diagnostic buffer
           43    13     0E63  4226          BEQL    BAD_PARAM               ; Br if none
00000000'GF    9F     0E65  4227          PUSHAB  G^EXE$READCHK           ; Set routine address for checking
           19    11     0E6B  4228          BRB     20$                     ; Join common code
                        0E6D  4229
                        0E6D  4230 CHECK_DGBUFR:                            ; Check read access to diag buffer
00000000'GF    9F     0E6D  4231          PUSHAB  G^EXE$WRITECHK          ; Set routine address to check buffer
           06    11     0E73  4232          BRB     10$                     ; Continue
                        0E75  4233
                        0E75  4234 CHECK_DGBUFW:                            ; Check write access to diag buffer
00000000'GF    9F     0E75  4235          PUSHAB  G^EXE$READCHK           ; Set routine to check write access to
                        0E7B  4236                                          ; buffer (no return no access)
                        0E7B  4237                                          ; Also sets IRP$V_FUNC in IRP
     50    6C    D0     0E7B  4238 10$:     MOVL    P1(AP),R0               ; Get address of diagnostic buffer
           28    13     0E7E  4239          BEQL    BAD_PARAM               ; Br if none
   51    04 AC    3C     0E80  4240          MOVZWL  P2(AP),R1               ; Get size of buffer
           22    13     0E84  4241          BEQL    BAD_PARAM               ; Br if none
           9E    16     0E86  4242 20$:     JSB     @(SP)+                  ; Call routine to perform checks
     52    50    D0     0E88  4243          MOVL    R0,R2                   ; Save address of user buffer
        FF7D    30     0E8B  4244          BSBW    ALLOC_P2BUF             ; Allocate a buffer
      1C 50    E9     0E8E  4245          BLBC    R0,ALLOC_FAIL           ; Br if allocation failure
   50    2C A3    D0     0E91  4246          MOVL    IRP$L_SVAPTE(R3),R0     ; Get system P2 buffer addr
   04 A0    52    D0     0E95  4247          MOVL    R2,P2B_L_BUFFER(R0)     ; Save user's P2 buffer addr
09 2A A3    01    E0     0E99  4248          BBS     #IRP$V_FUNC,IRP$W_STS(R3),30$ ; Br if READ function
                        0E9E  4249 ;
```

```
                                0E9E  4250 ; Move buffer to system buffer, if WRITE function
                                0E9E  4251 ;
                          38    BB  0E9E  4252          PUSHR   #^M<R3,R4,R5>           ; Save sacred registers
                    0C A0 62    51  28  0EA0  4253          MOVC3   R1,(R2),P2B_T_DATA(R0)  ; Save P2 char buffer
                          38    BA  0EA5  4254          POPR    #^M<R3,R4,R5>           ; Restore registers
                          05    05  0EA7  4255 30$:     RSB                             ; Return to checker routine
                                0EA8  4256
                                0EA8  4257 BAD_PARAM:                                 ; Bad parameter
                       50 14    9A  0EA8  4258          MOVZBL  #SS$_BADPARAM,R0        ; Set failure code
                          05    11  0EAB  4259          BRB     40$                     ;
                                0EAD  4260 ALLOC_FAIL:                                ; Allocation failure
                    50    0124 8F    3C  0EAD  4261          MOVZWL  #SS$_INSFMEM,R0         ; Set failure code
                             F4CE    31  0EB2  4262 40$:     BRW     ABORTIO                 ; Abort the I/O request
                                0EB5  4263          .DSABL  LSB
```

```
                        0EB5  4265          .SBTTL  STARTIO - START I/O OPERATION
                        0EB5  4266 ;++
                        0EB5  4267 ; STARTIO - START I/O OPERATION
                        0EB5  4268 ;
                        0EB5  4269 ; Functional description:
                        0EB5  4270 ;
                        0EB5  4271 ; This routine is called when an IRP is ready to be processed by the driver.
                        0EB5  4272 ; The request is dispatched to the appropriate routine base on the internal
                        0EB5  4273 ; function code in the IRP.
                        0EB5  4274 ;
                        0EB5  4275 ; Inputs:
                        0EB5  4276 ;
                        0EB5  4277 ;       R3 = IRP address
                        0EB5  4278 ;       R5 = UCB address
                        0EB5  4279 ;
                        0EB5  4280 ;       IPL = FIPL
                        0EB5  4281 ;
                        0EB5  4282 ; Outputs:
                        0EB5  4283 ;
                        0EB5  4284 ;       R0-R2,R4 are destroyed.
                        0EB5  4285 ;--
                        0EB5  4286
                        0EB5  4287 STARTIO::                                    ; Process an I/O packet
    51    21 A3   9A    0EB5  4288          MOVZBL  IRP$B_XE_FUNC(R3),R1        ; Get the internal function code
                        0EB9  4289 10$:     $DISPATCH       R1,TYPE=B,-
                        0EB9  4290          <-      ;function       action
                        0EB9  4291
                        0EB9  4292                  <XE_FC_V_INIT   STARTUP>,-      ; Startup request
                        0EB9  4293                  <XE_FC_V_RDCNTS INPUT>,-        ; Read counters request
                        0EB9  4294                  <XE_FC_V_RCCNTS INPUT>,-        ; Read and clear counters
                        0EB9  4295                  <XE_FC_V_STOP   SHUT>,-         ; Shutdown request
                        0EB9  4296                  <XE_FC_V_MULTI  INPUT>,-        ; Set new multicast list
                        0EB9  4297                  <XE_FC_V_LDMEM  INPUT_BUFFER1>,-; Load DEUNA/DELUA memory
                        0EB9  4298                  <XE_FC_V_RDMEM  INPUT_BUFFER>,- ; Read DEUNA/DELUA memory
                        0EB9  4299                  <XE_FC_V_UPST   INPUT>,-        ; Load and start Micro-address
                        0EB9  4300                  <XE_FC_V_LDSID  INPUT>,-        ; Load system ID parameters
                        0EB9  4301                  <XE_FC_V_RDSID  INPUT>,-        ; Read system ID parameters
                        0EB9  4302                  <XE_FC_V_STOPPR INPUT>,-        ; Stop xmit and receive process
                        0EB9  4303                  <XE_FC_V_STEST  INPUT>,-        ; Perform DEUNA/DELUA self test
                        0EB9  4304                  <XE_FC_V_READST INPUT>,-        ; Read port status
                        0EB9  4305                  <XE_FC_V_ESTEST INPUT>,-        ; Perform DEULA extended self test
                        0EB9  4306                  <XE_FC_V_HALT   INPUT>,-        ; Halt port (DELUA only)
                        0EB9  4307          >
                        0EF5  4308 ;
                        0EF5  4309 ; Other request type
                        0EF5  4310 ;
                        0EF5  4311          BUG_CHECK NOBUFPCKT,FATAL       ; Fatal error
                        0EF9  4312
                        0EF9  4313 ;
                        0EF9  4314 ; Startup unit's protocol
                        0EF9  4315 ;
                        0EF9  4316 STARTUP:                                     ; Startup unit's protocol
    45    10            0EF9  4317          BSBB    START                       ; Start protocol
01  50    E9            0EFB  4318          BLBC    R0,10$                      ; Br if error on startup
          05            0EFE  4319          RSB                                 ; Else return to caller
                        0EFF  4320
    50    DD            0EFF  4321 10$:     PUSHL   R0                          ; Save error return
```

```
                35   10  0F01  4322          BSBB    STOP                        ; Shutdown unit
                50 8ED0  0F03  4323          POPL    R0                          ; Restore error return
                         0F06  4324          ;BRB    DRV_DONE                    ; Complete request in error
                         0F06  4325
                         0F06  4326 DRV_DONE:
  3A A3         50   B0  0F06  4327          MOVW    R0,IRP$W_XE_STATUS(R3)      ; Set completion status
00B0 D4         63   0E  0F0A  4328          INSQUE  (R3),@CDB_Q_POST+4(R4)      ; Insert IRP on posting queue
          09F0        31  0F0F  4329          BRW     SCHED_FORKC                 ; Schedule a fork process
                         0F12  4330 ;
                         0F12  4331 ; Shutdown UNIT's protocol
                         0F12  4332 ;
                         0F12  4333 SHUT:                                        ; Shutdown protocol
54    0180 C5    D0  0F12  4334          MOVL    UCB$L_XE_CDB(R5),R4         ; Get CDB address
0224 C4    55   D1  0F17  4335          CMPL    R5,CDB_L_PRMUSER(R4)        ; Are we the PROMISCUOUS user?
          12   12  0F1C  4336          BNEQ    10$                         ; Br if not
     0224 C4   D4  0F1E  4337          CLRL    CDB_L_PRMUSER(R4)           ; Else, clear the PROMISCUOUS user
                         0F22  4338          CLRBIT  #PB_MOD_V_PROM,-            ; Reset the hardware mode
                         0F22  4339                  CDB_W_MODE(R4)              ;  out of promiscuous mode
          01   90  0F28  4340          MOVB    #NMA$C_STATE_OFF,-          ; Don't forget about the CDB
     025D C4         0F2A  4341                  CDB_B_PRM(R4)               ;  parameter
          03CE        31  0F2D  4342          BRW     INPUT                       ; Insert request on Input queue
                         0F30  4343
          06   10  0F30  4344 10$:    BSBB    STOP                        ; Shutdown unit
     50   01   9A  0F32  4345          MOVZBL  S^#SS$_NORMAL,R0            ; Return success
          1013        31  0F35  4346          BRW     IO_DONE                     ; Complete I/O request
                         0F38  4347 ;
                         0F38  4348 ; STOP the unit
                         0F38  4349 ;
                         0F38  4350 STOP:                                        ; Stop the protocol
54    0180 C5    D0  0F38  4351          MOVL    UCB$L_XE_CDB(R5),R4         ; Get CDB address
          15FA        31  0F3D  4352          BRW     SHUTDOWN_PROTYP             ; Shutdown the unit
```

```
                        0F40  4354          .SBTTL  START - START UNIT'S PROTOCOL
                        0F40  4355 ;++
                        0F40  4356 ; START - START UNIT'S PROTOCOL
                        0F40  4357 ;
                        0F40  4358 ; Functional description:
                        0F40  4359 ;
                        0F40  4360 ; This routine initiates the protocol on the unit. The DEUNA/DELUA is reset if
                        0F40  4361 ; first unit online.  The free list is filled and the first receive started.
                        0F40  4362 ; If a failure occurs the unit shutdown sequence is entered.
                        0F40  4363 ;
                        0F40  4364 ; Inputs:
                        0F40  4365 ;
                        0F40  4366 ;       R3 = IRP address
                        0F40  4367 ;       R5 = UCB address
                        0F40  4368 ;
                        0F40  4369 ;       IPL = FIPL
                        0F40  4370 ;
                        0F40  4371 ; Implicit inputs:
                        0F40  4372 ;
                        0F40  4373 ;       IRP$L_MEDIA contains a copy of the mode buffer specified by the user.
                        0F40  4374 ;       IRP$W_BOFF contains the quota taken from the user for the unit.
                        0F40  4375 ;
                        0F40  4376 ; Outputs:
                        0F40  4377 ;
                        0F40  4378 ;       R0 = Status return for startup request.
                        0F40  4379 ;       R1,R2,R4 are destroyed.
                        0F40  4380 ;       R3,R5 are preserved.
                        0F40  4381 ;
                        0F40  4382 ;--
                        0F40  4383
                        0F40  4384 START::                                      ; Start protocol operation
          00    91      0F40  4385          CMPB    #NMA$C_LINPR_POI,-          ;% Are we in PT-TO-PT mode?
00D8 C5                 0F42  4386                  UCB$B_XE_PRO(R5)            ;%
          24    12      0F45  4387          BNEQ    3$                          ;% Br if not
51  00C4 8F    3C      0F47  4388          MOVZWL  #IRP$C_LENGTH,R1            ;% Set size of an IRP
          53    DD      0F4C  4389          PUSHL   R3                          ;% Save R3
 00000000'GF    16      0F4E  4390          JSB     G^EXE$ALONONPAGED           ;% Allocate the IRP
          53 8ED0       0F54  4391          POPL    R3                          ;% Restore R3
       06 50    E8      0F57  4392          BLBS    R0,1$                       ;% Br if success
50  0124 8F    3C      0F5A  4393          MOVZWL  #SS$_INSFMEM,R0             ;% Set error return
          05    0F5F  4394          RSB                                 ;% Return to caller
                        0F60  4395
017C C5   52    D0      0F60  4396 1$:      MOVL    R2,UCB$L_XE_STIRP(R5)       ;% Save startup IRP
      0060 8F   A8      0F65  4397          BISW    #UCB$M_XE_START!UCB$M_XE_STACK,- ;% We are now in the startup
        68 A5           0F69  4398                  UCB$W_DEVSTS(R5)            ;%   and stack wait state
                        0F6B  4399
      00BC C5   D5      0F6B  4400 3$:      TSTL    UCB$L_XE_CPID(R5)           ; Creator PID saved already?
          06    12      0F6F  4401          BNEQ    5$                          ; Br if yes
00BC C5   20 A5   D0      0F71  4402          MOVL    UCB$L_CPID(R5),UCB$L_XE_CPID(R5) ; Else, save creator PID
                        0F77  4403 ;
                        0F77  4404 ; Set up idle UCB
                        0F77  4405 ;
      0800 8F   3C      0F77  4406 5$:      MOVZWL  #XM$M_STS_ACTIVE,-
        44 A5           0F7B  4407                  UCB$L_DEVDEPEND(R5)         ; Reset status and error summary
     21 A3    17    91      0F7D  4408      .   CMPB    #XE_FC_V_RESTART,IRP$B_XE_FUNC(R3) ; Is this a re-start operation?
          06    13      0F81  4409          BEQL    8$                          ; Br if yes - don't reset the PID
0088 C5   0C A3   D0      0F83  4410          MOVL    IRP$L_PID(R3),UCB$L_XE_PID(R5) ; Save starter's PID
```

```
                        0F89  4411 ;
                        0F89  4412 ; Check for CDB
                        0F89  4413 ;
52   0180 C5    D0      0F89  4414 8$:      MOVL    UCB$L_XE_CDB(R5),R2      ; Get CDB address, crash if not present
     0275 C2    96      0F8E  4415          INCB    CDB_B_UNTCNT(R2)         ; One more unit on this controller
          00    E0      0F92  4416          BBS     #CDB_STS_V_INITED,-      ; Br if already inited
  03 025A C2            0F94  4417                  CDB_B_STS(R2),10$        ;
        0090    31      0F98  4418          BRW     20$                      ; Else, init CDB
                        0F9B  4419
     54   52    D0      0F9B  4420 10$:     MOVL    R2,R4                    ; Copy CDB address
50   00DD C5    9E      0F9E  4421          MOVAB   UCB$B_XE_CDBPRM(R5),R0   ; Get UCB parameter address
51   025F C4    9E      0FA3  4422          MOVAB   CDB_B_SETPRM(R4),R1      ; Get CDB parameter address
     52   04    9A      0FA8  4423          MOVZBL  #UCB$C_XE_CDBPRM,R2      ; Set size of parameter list
                        0FAB  4424
                        0FAB  4425 ;
                        0FAB  4426 ; Check order of UCB parameters
                        0FAB  4427 ;
                        0FAB  4428          ASSUME  UCB$B_XE_CRC EQ UCB$B_XE_CDBPRM
                        0FAB  4429          ASSUME  UCB$B_XE_CON EQ UCB$B_XE_CRC+1
                        0FAB  4430          ASSUME  UCB$B_XE_EKO EQ UCB$B_XE_CON+1
                        0FAB  4431          ASSUME  UCB$B_XE_ILP EQ UCB$B_XE_EKO+1
                        0FAB  443^ ;
                        0FAB  4433 ; Check order of CDB parameters
                        0FAB  4434 ;
                        0FAB  4435          ASSUME  CDB_B_CRC EQ CDB_B_SETPRM
                        0FAB  4436          ASSUME  CDB_B_CON EQ CDB_B_CRC+1
                        0FAB  4437          ASSUME  CDB_B_EKO EQ CDB_B_CON+1
                        0FAB  4438          ASSUME  CDB_B_ILP EQ CDB_B_EKO+1
                        0FAB  4439
                        0FAB  4440          ASSUME  NMA$C_STATE_ON NE -1
                        0FAB  4441          ASSUME  NMA$C_STATE_OFF NE -1
     61   60    91      0FAB  4442 13$:     CMPB    (R0),(R1)                ; Match?
          70    12      0FAE  4443          BNEQ    18$                      ; Br if no
     80   81    90      0FB0  4444          MOVB    (R1)+,(R0)+              ; Store CDB value in UCB
       F5 52    F5      0FB3  4445          SOBGTR  R2,13$                   ; Loop if more to check
                        0FB6  4446 ;
                        0FB6  4447 ; NOW, check if user has given a hardware physical address.
                        0FB6  4448 ;
                        0FB6  4449 ;        R0 = Address of parameters in UCB
                        0FB6  4450 ;        R1 = Address of parameters in CDB
                        0FB6  4451 ;
                        0FB6  4452          ASSUME  CDB_G_PHA EQ CDB_B_ILP+1
                        0FB6  4453          ASSUME  UCB$G_XE_PHA EQ UCB$B_XE_ILP+1
60   FFFFFFFF 8F    D1      0FB6  4454          CMPL    #-1,(R0)                 ; Is user physical address defined?
          08    12      0FBD  4455          BNEQ    15$                      ; Br if yes
 04 A0    FFFF 8F    B1      0FBF  4456          CMPW    #-1,4(R0)                ; Is user physical address defined?
          10    13      0FC5  4457          BEQL    16$                      ; Br if not
 40 A3    0B04 8F    B0      0FC7  4458 15$:     MOVW    #NMA$C_PCLI_PHA,IRP$W_XE_CODE(R3) ; Assume bad physical address
     81   80    D1      0FCD  4459          CMPL    (R0)+,(R1)+              ; Physical address match??
          55    12      0FD0  4460          BNEQ    19$                      ; Br if no
     81   80    B1      0FD2  4461          CMPW    (R0)+,(R1)+              ; Still match??
          50    12      0FD5  4462          BNEQ    19$                      ; Br if no
     0263 C4    D0      0FD7  4463 16$:     MOVL    CDB_G_PHA(R4),-          ; Return hardware set address
     00E1 C5            0FDB  4464                  UCB$G_XE_PHA(R5)         ;  just in case we defaulted
     0267 C4    B0      0FDE  4465          MOVW    CDB_G_PHA+4(R4),-        ;       ...
     00E5 C5            0FE2  4466                  UCB$G_XE_PHA+4(R5)       ;       ...
                        0FE5  4467 ;
```

```
                                0FE5  4468 ; Check users buffer size - must not be more than twice the hardware buffer
                                0FE5  4469 ; size. (Already has been checked against max message size).
                                0FE5  4470 ;
      40 A3    0AF1 8F    B0    0FE5  4471          MOVW    #NMA$C_PCLI_BUS,IRP$W_XE_CODE(R3) ; Assume bad buffer size
         51    00B8 C4    3C    0FEB  4472          MOVZWL  CDB_W_BSZ(R4),R1          ; Get device buffer size
                                0FF0  4473          ASSUME  NMA$C_STATE_ON EQ 0
                                0FF0  4474          ASSUME  NMA$C_STATE_OFF EQ 1
            0A 025C C4    E8    0FF0  4475          BLBS    CDB_B_DCH(R4),17$         ; Br if data chaining OFF on DEUNA/DELUA
            05 00DA C5    E8    0FF5  4476          BLBS    UCB$B_XE_DCH(R5),17$      ; Br if user can't do data chaining
                                0FFA  4477                                            ; && Maybe this is an Internal IRP user
         51    00B8 C4    A0    0FFA  4478          ADDW    CDB_W_BSZ(R4),R1          ; Compute twice the normal buffer size
            51    42 A5    B1    0FFF  4479 17$:     CMPW    UCB$W_DEVBUFSIZ(R5),R1    ; Is buffer size okay?
                     22    1A    1003  4480          BGTRU   19$                       ; Br if too large
                  40 A3    B4    1005  4481          CLRW    IRP$W_XE_CODE(R3)         ; No bad parameters
                     10    A8    1008  4482          BISW    #UCB$M_XE_RUN,-           ; Indicate we have entered RUN mode
                  68 A5          100A  4483                  UCB$W_DEVSTS(R5)          ;
            21 A3    18    90    100C  4484          MOVB    #XE_FC_V_CHMODE,IRP$B_XE_FUNC(R3) ; Set new function code
                                1010  4485          SETIPL  UCB$B_DIPL(R5)            ; Sync access to UCB and CDB
         0088 D4    63    0E    1014  4486          INSQUE  (R3),@CDB_Q_INPUT+4(R4)   ; Insert at END of input queue
                 04A0    30    1019  4487          BSBW    LOAD_PORT                 ; Load the DEUNA/DELUA port
               50    01    9A    101C  4488          MOVZBL  #SS$_NORMAL,R0            ; Return success
                           05    101F  4489          RSB                               ; Return to caller
                                1020  4490 ;
                                1020  4491 ; Error on parameter validation
                                1020  4492 ;
      40 A3    F184 CF42    B0    1020  4493 18$:     MOVW    BAD_PARAM_TBL-2[R2],IRP$W_XE_CODE(R3) ; Return parameter code
               50    14    9A    1027  4494 19$:     MOVZBL  S^#SS$_BADPARAM,R0        ; Return bad parameter error
                           05    102A  4495          RSB                               ; Return to caller
                                102B  4496 ;
                                102B  4497 ; Initialize CDB
                                102B  4498 ;
               51    0222 8F    3C    102B  4499 20$:     MOVZWL  #CDB_C_ZERO,R1            ; Get portion of CDB to init with zero
                           3E    BB    1030  4500          PUSHR   #^M<R1,R2,R3,R4,R5>       ; Save registers
   62    51    00    62    00    2C    1032  4501          MOVC5   #0,(R2),#0,R1,(R2)        ; Zero the structure
                           3E    BA    1038  4502          POPR    #^M<R1,R2,R3,R4,R5>       ; Restore registers
                                103A  4503          ASSUME  CDB_L_FQFL EQ 0
                                103A  4504          ASSUME  CDB_L_FQBL EQ CDB_L_FQFL+4
                     54    82    7E    103A  4505          MOVAQ   (R2)+,R4                  ; Skip link pointers, copy CDB address
                                103D  4506          ASSUME  CDB_W_SIZE EQ CDB_L_FQBL+4
                     82    51    B0    103D  4507          MOVW    R1,(R2)+                  ; Store size of structure
                                1040  4508          ASSUME  CDB_B_TYPE EQ CDB_W_SIZE+2
                                1040  4509          ASSUME  CDB_B_FIPL EQ CDB_B_TYPE+1
               82    0833 8F    B0    1040  4510          MOVW    #<IPL$_XE_FIPL@8>!DYN$C_CDB,(R2)+; Set structure type and FIPL
                                1045  4511          ASSUME  CDB_L_FPC EQ CDB_B_FIPL+1
         82    00001921'EF    9E    1045  4512          MOVAB   FORK_PROC,(R2)+           ; Set fork process address
                                104C  4513          ASSUME  CDB_L_FR3 EQ CDB_L_FPC+4
                                104C  4514          ASSUME  CDB_L_FR4 EQ CDB_L_FR3+4
                     82    7C    104C  4515          CLRQ    (R2)+                     ; Clear fork R3 and R4
                                104E  4516          ASSUME  CDB_B_NEXTXMT EQ CDB_L_FR4+4
                                104E  4517          ASSUME  CDB_B_NEXTRCV EQ CDB_B_NEXTXMT+1
                                104E  4518          ASSUME  CDB_B_RCVMAP EQ CDB_B_NEXTRCV+1
                                104E  4519          ASSUME  CDB_B_XMTMAP EQ CDB_B_RCVMAP+1
                     82    D4    104E  4520          CLRL    (R2)+                     ; Clear slot in use flags
                                1050  4521          ASSUME  CDB_L_RCVMAP EQ CDB_B_XMTMAP+1
                  51    0C    9A    1050  4522          MOVZBL  #MAX_C_XMT+MAX_C_RCV,R1   ; Set number of mapping vectors
                  82    01    CE    1053  4523 30$:     MNEGL   #1,(R2)+                  ; Indicate no mapping info
                     FA 51    F5    1056  4524          SOBGTR  R1,30$                    ; Loop if more
```

```
                                1059  4525          ASSUME  CDB_L_XRADDR EQ CDB_L_RCVMAP+<4*<MAX_C_XMT+MAX_C_RCV>>
                                1059  4526          ASSUME  CDB_L_RRADDR EQ CDB_L_XRADDR+<4*MAX_C_XMT>
                                1059  4527          ASSUME  CDB_G_RRING  EQ CDB_G_XRING+XMT_K_LENGTH
                                1059  4528          ASSUME  RCV_C_LENGTH EQ XMT_C_LENGTH
        51    0C    9A        1059  4529          MOVZBL  #MAX_C_XMT+MAX_C_RCV,R1 ; Set total number of ring entries
50      01AA C4     9E        105C  4530          MOVAB   CDB_G_XRING(R4),R0      ; Set address of first ring entry
        82    50    D0        1061  4531 35$:     MOVL    R0,(R2)+                ; Store ring entry address in table
        50    0A    C0        1064  4532          ADDL2   #RCV_C_LENGTH,R0        ; Compute address of next ring entry
          F7  51    F5        1067  4533          SOBGTR  R1,35$                  ; Loop if more
                                106A  4534          ASSUME  CDB_Q_QUEUES EQ CDB_L_XRADDR+<4*<MAX_C_XMT+MAX_C_RCV>>
        51    07    9A        106A  4535          MOVZBL  #CDB_C_QUEUES,R1        ; Set number of queue listheads
        82    62    DE        106D  4536 40$:     MOVAL   (R2),(R2)+              ; Set forward link
  82      FC  A2    D0        1070  4537          MOVL    -4(R2),(R2)+            ; Set backward link
          F6  51    F5        1074  4538          SOBGTR  R1,40$                  ; Loop if more listheads
                                1077  4539          SETBIT  #CDB_STS_V_INITED,CDB_B_STS(R4) ; Set initial status bits
00BC C4     01    CE          107D  4540          MNEGL   #1,CDB_L_PCBBMAP(R4)    ; Set no mapping for PCBB yet
                                1082  4541
                                1082  4542 ;
                                1082  4543 ; Because the recieve buffer sizes are a constant, the buffer length fields of
                                1082  4544 ; the ring descriptor entries can be setup only once. Note that the DEUNA/DELUA
                                1082  4545 ; receive buffer size must be a multiple of 2
                                1082  4546 ;
                                1082  4547
                                1082  4548          ASSUME  <XE_C_HEADER+XE_C_CRC+XE_C_CNTSIZ & 1 > EQ 0
        51    08    9A        1082  4549          MOVZBL  #MAX_C_RCV,R1           ; Set total number of ring entries
50      01D2 C4     9E        1085  4550          MOVAB   CDB_G_RRING(R4),R0      ; Set address of first ring entry
              A1    1A        108A  4551          ADDW3   #XE_C_HEADER+XE_C_CRC-  ; Calculate buffer length
                              108B  4552                  +XE_C_CNTSIZ,-          ;       ...
52      00D6 C5     14        108B  4553                  UCB$W_XE_BSZ(R5),R2     ;       ...
        60    52    B0        1090  4554 43$:     MOVW    R2,RCV_W_SLEN(R0)       ; Store buffer length in current entry
        50    0A    C0        1093  4555          ADDL2   #RCV_C_LENGTH,R0        ; Compute address of next ring entry
          F7  51    F5        1096  4556          SOBGTR  R1,43$                  ; Loop if more
                                1099  4557
                                1099  4558 ;
                                1099  4559 ; Allocate map registers for receive buffers and one transmit buffer. The
                                1099  4560 ; any datapath (buffered/unbuffered) can be used for all I/O's due to the
                                1099  4561 ; fact that all data is buffered on the DEUNA/DELUA.
                                1099  4562 ;
                                1099  4563 ; We will allocate the buffered data paths, only as needed. No buffered
                                1099  4564 ; data paths are pre-allocated. Also, we can only use the buffered data
                                1099  4565 ; paths for transmit operations. This is because receives have no guaranteed
                                1099  4566 ; completion time and we do not want to waste the buffered data paths!
                                1099  4567 ;
                                1099  4568
        00D6 C5     B0        1099  4569          MOVW    UCB$W_XE_BSZ(R5),-      ; Init CDB buffer size
        00B8 C4               109D  4570                  CDB_W_BSZ(R4)           ;
00BA C4 00D2 C5     B0        10A0  4571          MOVW    UCB$W_XE_HBQ(R5),CDB_W_QUOTA(R4) ; Set initial quota
                              10A7  4572
                              10A7  4573          ASSUME  INIT_C_AQUOTA LE 255
                              10A7  4574          ASSUME  CDB_B_AQUOTA EQ CDB_B_AQUOTA+1
        0200 8F     B0        10A7  4575          MOVW    #<INIT_C_AQUOTA@8>,-    ; Initialize Maximum QUOTA
        02B8 C4               10AB  4576           CDB_B_AQUOTA(R4)               ; and zero Additional QUOTA
                              10AE  4577          ASSUME  UCB$B_XE_MLT EQ UCB$B_XE_PRM+1
                              10AE  4578          ASSUME  CDB_B_MLT EQ CDB_B_PRM+1
025D C4 00DB C5     B0        10AE  4579          MOVW    UCB$B_XE_PRM(R5),CDB_B_PRM(R4) ; Set the promiscuous mode
                              10B5  4580                                          ;  and the all multicast enable
    50  00DD C5     9E        10B5  4581          MOVAB   UCB$B_XE_CDBPRM(R5),R0  ; Get address of UCB parameters
```

```
                51    025F C4    9E  10BA  4582          MOVAB   CDB_B_SETPRM(R4),R1      ; Get address of CDB parameters
                      52    0A   9A  10BF  4583          MOVZBL  #UCB$C_XE_SETPRM,R2     ; Set size of parameter list
                      81    80   90  10C2  4584 45$:     MOVB    (R0)+,(R1)+             ; Store parameters
                        FA 52    F5  10C5  4585          SOBGTR  R2,45$                  ; Loop if more
                         1A74    30  10C8  4586          BSBW    MOVE_MULTI              ; Copy multicast address list
                      00B8 C4    B0  10CB  4587          MOVW    CDB_W_BSZ(R4),-         ; Set buffer size
                        7E A5          10CF  4588                  UCB$W_BCNT(R5)
             7C A5    01FF 8F    B0  10D1  4589          MOVW    #511,UCB$W_BOFF(R5)     ; Set worst case byte offset
                                10D7  4590          PUSHQ   R6                      ; Save R6,R7
                      56    54   D0  10DA  4591          MOVL    R4,R6                   ; Save CDB address
                54    24 A5    D0  10DD  4592          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
                                10E1  4593          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                10E1  4594          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                      34 A4    D4   0E1  4595          CLRL    CRB$L_INTD+VEC$W_MAPREG(R4) ; Clear map register + datapath
                56    1C A6    DE  10E4  4596          MOVAL   CDB_L_RCVMAP(R6),R6     ; Get mapping slot address
                   57    08    9A  10E8  4597          MOVZBL  #MAX_C_RCV,R7           ; Get number of receive slots
                00000000'GF    16  10EB  4598 50$:     JSB     G^IOC$ALOUBAMAP         ; Allocate a set of map registers
                      43 50    E9  10F1  4599          BLBC    R0,70$                  ; Br if unavailable
                86    34 A4    D0  10F4  4600          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),(R6)+ ; Save map info
                      F0 57    F5  10F8  4601          SOBGTR  R7,50$                  ; Continue
                                10FB  4602
                   05EE 8F    3C  10FB  4603          MOVZWL  #MAX_PKT_SIZE+18,-      ; Set transmit buffer size to max
                        7E A5       10FF  4604                  UCB$W_BCNT(R5)          ;  Ethernet packet size + header
                00000000'GF    16  1101  4605 60$:     JSB     G^IOC$ALOUBAMAP         ; Allocate a set of transmit registers
                      2D 50    E9  1107  4606          BLBC    R0,70$                  ; Br if unavailable
                                110A  4607          ASSUME  CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*<MAX_C_RCV>>
                66    34 A4    D0  110A  4608          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),(R6) ; Save transmit map info
                                110E  4609 ;
                                110E  4610 ; Calculate UNIBUS mapping for PCBB and other DEUNA/DELUA structures.
                                110E  4611 ;
                56    10 A4    D0  110E  4612          MOVL    CRB$L_AUXSTRUC(R4),R6   ; Get CDB address
                57    28 A5    D0  1112  4613          MOVL    UCB$L_DDB(R5),R7        ; Get DDB address
             00C8 C6    04 A7    D0  1116  4614          MOVL    DDB$L_UCB(R7),CDB_L_UCB0(R6) ; Set UCB #0 address
                57    00D6 C6    9E  111C  4615          MOVAB   CDB_G_PCBB(R6),R7       ; Get PCBB address
                      FE00 8F    AB  1121  4616          BICW3   #^C<VA$M_BYTE>,-        ; Get PCBB byte offset
                   7C A5    57       1125  4617                  R7,UCB$W_BOFF(R5)
             7E A5    014C 8F    B0  1128  4618          MOVW    #CDB_C_MAPPED,UCB$W_BCNT(R5) ; Set Block size
                00000000'GF    16  112E  4619          JSB     G^IOC$ALOUBAMAP         ; Allocate map registers
                         09 50    E8  1134  4620          BLBS    R0,80$                  ; Br if allocated
                                1137  4621 7U$:     POPQ    R6                      ; Restore R6, R7
                50    0344 8F    3C  113A  4622          MOVZWL  #SS$_INSFMAPREG,R0      ; Set insufficient map registers
                                05  113F  4623          RSB                             ; Return with error
                                1140  4624
                                1140  4625          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                1140  4626          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                      34 A4    D0  1140  4627 80$:     MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Save PCBB mapping info
                   00BC C6       1143  4628                  CDB_L_PCBBMAP(R6)       ;
                                1146  4629 ;
                                1146  4630 ; Convert virtual PCB address to physical PTE address
                                1146  4631 ;
                         09    EF  1146  4632          EXTZV   S^#VA$V_VPN,-           ; Get PCBB page number
                51    57    15       1148  4633                  S^#VA$S_VPN,R7,R1       ;
             50    00000000'GF    D0  114B  4634          MOVL    G^MMG$GL_SPTBASE,R0     ; Get SPT address
                78 A5    6041    DE  1152  4635          MOVAL   (R0)[R1],UCB$L_SVAPTE(R5) ; Set PTE address
                00000000'GF    16  1157  4636          JSB     G^IOC$LOADUBAMAP        ; Load the PCBB map registers
                      02    07    EF  115D  4637          EXTZV   #7,#2,-                 ; Get BA16-BA17
                  51    34 A4       1160  4638                  CRB$L_INTD+VEC$W_MAPREG(R4),R1 ;
```

```
                51    51    10    78  1163  4639          ASHL    #16,R1,R1                ; Move to high word
                  51    7C A5    B0  1167  4640          MOVW    UCB$W_BOFF(R5),R1        ; Set BA0-BA8
                        34 A4    F0  116B  4641          INSV    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Set BA9-BA15
                51    07    09        116E  4642                  #9,#7,R1                 ;
                00C0 C6     51    D0  1171  4643          MOVL    R1,CDB_L_PCBBUA(R6)      ; Save PCBB UNIBUS address
                                      1176  4644 ;
                                      1176  4645 ; Initialize ring buffer descriptor
                                      1176  4646 ;
             51   000000D4 8F    C0  1176  4647          ADDL    #CDB_G_XRING-CDB_G_PCBB,R1 ; Get address of XMIT RING
                                      117D  4648          ASSUME  UDB_BUF_W_TDRBL EQ 0
                00DE C6     51    D0  117D  4649          MOVL    R1,CDB_G_XRUDB(R6)       ; Store XMIT RING UNIBUS address
                            05    90  1182  4650          MOVB    #<XMT_C_LENGTH/2>,-      ; Store size of XMIT RING entry in words
                      00E1 C6       1184  4651                  CDB_G_XRUDB+UDB_BUF_B_TELEN(R6) ;
                            04    9B  1187  4652          MOVZBW  #MAX_C_XMT,-             ; Store number of ring entries
                      00E2 C6       1189  4653                  CDB_G_XRUDB+UDB_BUF_W_TRLEN(R6) ;
                51    28    C0  118C  4654          ADDL    #CDB_G_RRING-CDB_G_XRING,R1 ; Get address of RECV RING
                00E4 C6     51    D0  118F  4655          MOVL    R1,CDB_G_XRUDB+UDB_BUF_W_RDRBL(R6) ; Store RCV RING BUS address
                            05    90  1194  4656          MOVB    #<RCV_C_LENGTH/2>,-      ; Store size of RECV RING entry in words
                      00E7 C6       1196  4657                  CDB_G_XRUDB+UDB_BUF_B_RELEN(R6) ;
                            08    9B  1199  4658          MOVZBW  #MAX_C_RCV,-             ; Store number of ring entries
                      00E8 C6       119B  4659                  CDB_G_XRUDB+UDB_BUF_W_RRLEN(R6) ;
                                      119E  4660 ;
                                      119E  4661 ; Initialize DEUNA/DELUA mode word
                                      119E  4662 ;
                                      119E  4663          ASSUME  NMA$C_STATE_ON EQ 0
                                      119E  4664          ASSUME  NMA$C_STATE_OFF EQ 1
          0258 C6   EEE2 CF    B0  119E  4665          MOVW    DEF_MODE,CDB_W_MODE(R6) ; Reset mode to initial value
               07 025D C6     E8  11A5  4666          BLBS    CDB_B_PRM(R6),90$        ; Br if promiscuous state is OFF
                  8000 8F     A8  11AA  4667          BISW    #PB_MOD_M_PROM,-         ; Else, enable promiscuous mode
                  0258 C6         11AE  4668                  CDB_W_MODE(R6)           ;
                                  11B1  4669
               07 025E C6     E8  11B1  4670 90$:     BLBS    CDB_B_MLT(R6),100$       ; Br if multicast state is OFF
                  4000 8F     A8  11B6  4671          BISW    #PB_MOD_M_ENAL,-         ; Else, enable all MULTICASTS
                  0258 C6         11BA  4672                  CDB_W_MODE(R6)           ;
                                  11BD  4673
          00001F9C'EF     9E    11BD  4674 100$:    MOVAB   ASSPKT_NODCH,-           ; Assume data chaining is disabled
                  02B4 C6         11C3  4675                  CDB_L_ASSPKT(R6)
               19 025C C6     E8  11C6  4676          BLBS    CDB_B_DCH(R6),110$       ; Branch if data chaining is OFF
                  05DC 8F     B1  11CB  4677          CMPW    #MAX_PKT_SIZE,-          ; Don't enable data chaining if the
                  00B8 C6         11CF  4678                  CDB_W_BSZ(R6)            ; recieve buffer size is greater than
                        10    1B  11D2  4679          BLEQU   110$                     ; or equal to the maximum packet size
                  2000 8F     AA  11D4  4680          BICW2   #PB_MOD_M_DRDC,-         ; Enable data chaining
                  0258 C6         11D8  4681                  CDB_W_MODE(R6)
          00001FC3'EF     9E    11DB  4682          MOVAB   ASSPKT_DCH,-             ; Use data chaining ASSPKT routine
                  02B4 C6         11E1  4683                  CDB_L_ASSPKT(R6)
                                  11E4  4684
               07 025B C6     E8  11E4  4685 110$:    BLBS    CDB_B_PAD(R6),120$       ; Br if padding is OFF
                  1000 8F     A8  11E9  4686          BISW    #PB_MOD_M_TPAD,-         ; Else, enable padding
                  0258 C6         11ED  4687                  CDB_W_MODE(R6)           ;
                                  11F0  4688
               05 025F C6     E9  11F0  4689 120$:    BLBC    CDB_B_CRC(R6),125$       ; Br if CRC is to be generated
                        08    A8  11F5  4690          BISW    #PB_MOD_M_DTCR,-         ; Else, disable CRC generation
                  0258 C6         11F7  4691                  CDB_W_MODE(R6)           ;
                                  11FA  4692
               25 0261 C6     E9  11FA  4693 125$:    BLBC    CDB_B_EKO(R6),130$       ; Br if Echo mode is enabled (ON)
               51   0A A5     9A  11FF  4694          MOVZBL  UCB$B_TYPE(R5),R1        ; Retrieve device type
                                  1203  4695          $DISPATCH       R1,TYPE=B,-      ; Dispatch based upon device type
```

```
                                1203  4696          < -
                                1203  4697                <DT$_DEUNA      127$>,- ; DEUNA - disable echo mode
                                1203  4698                <DT$_DELUA      130$>,- ; DELUA - echo mode is always disabled
                                1203  4699          >
                01   A8  121F  4700 127$:   BISW    #PB_MOD_M_HDX,-         ; Else, disable Echo mode - set to HDX
           0258 C6          1221  4701                CDB_W_MODE(R6)          ;
                            1224  4702
                            1224  4703 130$:   ASSUME  NMA$C_LINCN_NOR EQ 0
                            1224  4704          ASSUME  NMA$C_LINCN_LOO EQ 1
      16 0260 C6   E9       1224  4705          BLBC    CDB_B_CON(R6),140$      ; Br if LOOPBACK is not enabled
                04   A8     1229  4706          BISW    #PB_MOD_M_LOOP,-        ; Else, enable loopback
           0258 C6          122B  4707                CDB_W_MODE(R6)          ;
                01   AA     122E  4708          BICW    #PB_MOD_M_HDX,-         ; Must set ourselves to FDX
           0258 C6          1230  4709                CDB_W_MODE(R6)          ;  ... what a KLUGE
                            1233  4710
      07 0262 C6   E8       1233  4711          BLBS    CDB_B_ILP(R6),140$      ;
           0040 8F   A8     1238  4712          BISW    #PB_MOD_M_INTL,-        ; Else, enable internal loopback (DELUA)
           0258 C6          123C  4713                CDB_W_MODE(R6)          ;
                            123F  4714
                54   56     D0    123F  4715 140$:   MOVL    R6,R4                   ; Set R4 to CDB address
                            1242  4716          POPQ    R6                      ; Restore R6, R7
                            1245  4717 ;
                            1245  4718 ; Setup fork process to start CDB timer
                            1245  4719 ;
                38   BB     1245  4720          PUSHR   #^M<R3,R4,R5>           ; Save registers
                04   E2     1247  4721          BBSS    #CDB_STS_V_TIMER,-      ; Br if timer already going
           24 025A C4       1249  4722                CDB_B_STS(R4),150$      ;
                04   88     124D  4723          BISB    #DPT$M_NOUNLOAD,-       ; Do not allow driver to be unloaded
      0000000D'EF           124F  4724                DPT$TAB+DPT$B_FLAGS     ;  while the TQE is active
           55   0228 C4     1254  4725          MOVAL   CDB_L_TQE(R4),R5        ; Get the TQE address
20 A5 00000000 01312D00 8F   1259  4726          MOVQ    #TQE_DELTA,TQE$Q_DELTA(R5) ; Set the delta time
           53   2298'CF     1265  4727          MOVAB   W^TQE_TIMER,R3          ; Set address of timer wakeup routine
             2C A5   05     126A  4728          MOVB    #TQE$C_SSREPT,TQE$L_RQPID(R5) ; Set the TQE request type
                   0050     126E  4729          BSBW    FORK_TIMER              ; Create fork process for TQE
                            1271  4730 ;
                            1271  4731 ; Put IRP on input queue to await master reset completion interrupt.
                            1271  4732 ;
                38   BA     1271  4733 150$:   POPR    #^M<R3,R4,R5>           ; Restore registers
           0084 C4   63     1273  4734          INSQUE  (R3),CDB_Q_INPUT(R4)    ; Insert at front of input queue
             52   24 A5     1278  4735          MOVL    UCB$L_CRB(R5),R2        ; Get CRB adddress
             52   2C B2     127C  4736          MOVL    @CRB$C_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
                            1280  4737 ;
                            1280  4738 ; Master reset device
                            1280  4739 ;
                            1280  4740          DSBINT  UCB$B_DIPL(R5)          ; Sync access to UCB
                62   20     1287  4741          MOVW    #XE_PCSR0_M_RSET,PCSR0(R2) ; Master Reset device
                            128A  4742 ;
                            128A  4743 ; The master reset will take some time to complete ...
                            128A  4744 ; so we will execute some NOPs to give the DEUNA/DELUA some time.
                            128A  4745 ;
                            128A  4746          TIMEWAIT #1,#XE_PCSR0_M_DNI,PCSR0(R2),W ; Wait for DNI or 10 usec
                50   01     12AF  4747          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
           62   0040 8F     12B2  4748          MOVW    #XE_PCSR0_M_INTE,PCSR0(R2) ; Enable interrupts
                            12B7  4749          WFIKPCH TIMEOUT,#T6             ; Wait for self test done interrupt.
                            12C1  4750                                          ; We will want all initialization
                            12C1  4751                                          ; to complete before clearing timeout
                            12C1  4752                                          ; and interrupt bits in UCB$W_STS(R5).
```

```
                        12C1  4753
                        12C1  4754 ;++
                        12C1  4755 ; FORK_TIMER - Routine to create fork a process to start a timer
                        12C1  4756 ;
                        12C1  4757 ; This routine starts up FORK process which is used to start a timer.
                        12C1  4758 ;
                        12C1  4759 ; Inputs:
                        12C1  4760 ;
                        12C1  4761 ;       R3 = Address of system routine to handle the timer expiration.
                        12C1  4762 ;       R5 = Address of TQE block
                        12C1  4763 ;
                        12C1  4764 ;       IPL = Greater than Queueast IPL
                        12C1  4765 ;
                        12C1  4766 ; Outputs:
                        12C1  4767 ;
                        12C1  4768 ;       Fork process is started.
                        12C1  4769 ;       R3, R4 are destroyed by EXE$FORK
                        12C1  4770 ;
                        12C1  4771 ;--
                        12C1  4772 FORK_TIMER:                                  ; Create fork process to start timer
 0A A5    0F    90      12C1  4773         MOVB    #DYN$C_TQE,FKB$B_TYPE(R5) ; Set structure type
 0B A5    06    90      12C5  4774         MOVB    #IPL$_QUEUEAST,FKB$B_FIPL(R5) ; Set IPL of fork process
       D2'AF    9F      12C9  4775         PUSHAB  B^START_TIMER             ; Push address of fork process
00000000'GF    17      12CC  4776         JMP     G^EXE$FORK                ; Create fork process to start timer
                        12D2  4777
                        12D2  4778 ;++
                        12D2  4779 ; START_TIMER - Fork process to start the CDB timer
                        12D2  4780 ;
                        12D2  4781 ; This routine starts up the CDB which is used to monitor the DEUNA/DELUA
                        12D2  4782 ; controller for proper operation.
                        12D2  4783 ;
                        12D2  4784 ; Inputs:
                        12D2  4785 ;
                        12D2  4786 ;       R5 = Address of TQE block
                        12D2  4787 ;
                        12D2  4788 ;       IPL = Queueast IPL
                        12D2  4789 ;
                        12D2  4790 ; Implicit inputs:
                        12D2  4791 ;
                        12D2  4792 ;       TQE$Q_DELTA(R5) = Delta time interval
                        12D2  4793 ;       TQE$L_RQPID(R5) = TQE request type (SSSNGL or SSREPT)
                        12D2  4794 ;
                        12D2  4795 ; Outputs:
                        12D2  4796 ;
                        12D2  4797 ;       R0-R3 are destroyed.
                        12D2  4798 ;       TQE element added to timer queue
                        12D2  4799 ;
                        12D2  4800 ;--
                        12D2  4801 START_TIMER:
                        12D2  4802         DSBINT  #IPL$_TIMER               ; Raise IPL
 0C A5    10 A5    D0    12D8  4803         MOVL    FKB$L_FR3(R5),TQE$L_FPC(R5) ; Set address of timer wakeup
          2C A5    90    12DD  4804         MOVB    TQE$L_RQPID(R5),-         ; Set TQE request type
          0B A5          12E0  4805                 TQE$B_RQTYPE(R5)          ;
    50    20 A5    7D    12E2  4806         MOVQ    TQE$Q_DELTA(R5),R0        ; Get delta time
50  00000000'GF    C0    12E6  4807         ADDL    G^EXE$GQ_SYSTIME,R0       ; Add in current time
51  00000004'GF    D8    12ED  4808         ADWC    G^EXE$GQ_SYSTIME+4,R1     ;       ...
    00000000'GF    16    12F4  4809         JSB     G^EXE$INSTIMQ             ; Insert element on timer queue
```

```
                12FA  4810          ENBINT
                                    ; Restore IPL
05              12FD  4811          RSB
```

```
                     12FE  4813          .SBTTL  INPUT - INSERT REQUEST ON INPUT PROCESS QUEUE
                     12FE  4814 ;++
                     12FE  4815 ; INPUT - INSERT REQUEST ON INPUT PROCESS QUEUE
                     12FE  4816 ;
                     12FE  4817 ; Functional description:
                     12FE  4818 ;
                     12FE  4819 ; This routine inserts the IRP packet on the input waiting queue and
                     12FE  4820 ; loads the port CSR's.
                     12FE  4821 ;
                     12FE  4822 ; Inputs:
                     12FE  4823 ;
                     12FE  4824 ;       R3 = IRP address
                     12FE  4825 ;       R5 = UCB address
                     12FE  4826 ;
                     12FE  4827 ;       IPL = FIPL
                     12FE  4828 ;
                     12FE  4829 ; Outputs:
                     12FE  4830 ;
                     12FE  4831 ;       The request is completed.
                     12FE  4832 ;--
                     12FE  4833 INPUT::                                      ; Insert request on input queue
                     12FE  4834          ASSUME  UCB$V_XE_INITED  EQ  0
    08 68 A5    E8   12FE  4835          BLBS    UCB$W_DEVSTS(R5),10$        ; Br if unit inited
50   20D4 8F    3C   1302  4836          MOVZWL  #SS$_DEVINACT,R0            ; Else, set error return
        0C41    31   1307  4837          BRW     IO_DONE                     ; Finish the request
                     130A  4838
54   0180 C5    D0   130A  4839 10$:     MOVL    UCB$L_XE_CDB(R5),R4         ; Get CDB address
                     130F  4840          SETIPL  UCB$B_DIPL(R5)              ; Sync access to device
0088 D4   63    0E   1313  4841          INSQUE  (R3),@CDB_Q_INPUT+4(R4)     ; Insert at END of queue
        01A1    31   1318  4842          BRW     LOAD_PORT
```

```
                     131B  4844          .SBTTL  INPUT_BUFFER - INSERT REQUEST ON INPUT PROCESS QUEUE AND MAP BUFFER
                     131B  4845 ;++
                     131B  4846 ; INPUT_BUFFER - INSERT REQUEST ON INPUT PROCESS QUEUE AND MAP BUFFER
                     131B  4847 ;
                     131B  4848 ; Functional description:
                     131B  4849 ;
                     131B  4850 ; This routine inserts the IRP packet on the input waiting queue and
                     131B  4851 ; loads the CSR's.
                     131B  4852 ;
                     131B  4853 ; Inputs:
                     131B  4854 ;
                     131B  4855 ;       R3 = IRP address
                     131B  4856 ;       R5 = UCB address
                     131B  4857 ;
                     131B  4858 ;       IPL = FIPL
                     131B  4859 ;
                     131B  4860 ; Outputs:
                     131B  4861 ;
                     131B  4862 ;       R0,R1,R2 are destroyed.
                     131B  4863 ;       The request is completed.
                     131B  4864 ;--
                     131B  4865          .ENABL  LSB
                     131B  4866 INPUT_BUFFER1::                              ; Insert request on input queue
54   0180 C5    D0   131B  4867          MOVL    UCB$L_XE_CDB(R5),R4        ; Get CDB address
             0F    13   1320  4868          BEQL    3$                         ; Br if no CDB
50   02C4 8F    3C   1322  4869          MOVZWL  #SS$_DEVACTIVE,R0          ; Assume unit is running
             01    E0   1327  4870          BBS     #CDB_STS_V_RUN,-           ; Br if DEUNA/DELUA is running - error
   09 025A C4         1329  4871                  CDB_B_STS(R4),5$           ;
                     132D  4872 INPUT_BUFFER::                               ; Insert request on input queue
                     132D  4873          ASSUME  UCB$V_XE_INITED  EQ  0
   08 68 A5    E8   132D  4874          BLBS    UCB$W_DEVSTS(R5),10$       ; Br if unit inited - okay
50    20D4 8F    3C   1331  4875 3$:     MOVZWL  #SS$_DEVINACT,R0           ; Else, set error return
        0C12    31   1336  4876 5$:     BRW     IO_DONE                    ; Finish the request
                     1339  4877
                     1339  4878          ASSUME  P2B_L_POINTER EQ 0
        56    DD   1339  4879 10$:    PUSHL   R6                         ; Save a register
                     133B  4880 ;
                     133B  4881 ; Set up variant mapping info in UCB and CRB to allocate a set of map
                     133B  4882 ; registers for the buffer.
                     133B  4883 ;
56    2C B3    D0   133B  4884          MOVL    @IRP$L_SVAPTE(R3),R6       ; Get buffer address
   FE00 8F    AB   133F  4885          BICW3   #^C<VA$M_BYTE>,-           ; Get byte offset
7C A5    56         1343  4886                  R6,UCB$W_BOFF(R5)          ;
      32 A3    B0   1346  4887          MOVW    IRP$W_BCNT(R3),-           ; Set Block size
      7E A5         1349  4888                  UCB$W_BCNT(R5)             ;
54    24 A5    D0   134B  4889          MOVL    UCB$L_CRB(R5),R4           ; Get CRB address
                     134F  4890          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                     134F  4891          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
      34 A4    D4   134F  4892          CLRL    CRB$L_INTD+VEC$W_MAPREG(R4) ; Reset mapping info
00000000'GF    16   1352  4893          JSB     G^IOC$ALOUBAMAP            ; Allocate map registers
      47 50    E9   1358  4894          BLBC    R0,20$                     ; Br if NOT allocated
                     135B  4895          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                     135B  4896          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
      34 A4    D0   135B  4897          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Save mapping info in IRP
      3C A3         135E  4898                  IRP$L_XE_MAP(R3)           ;
                     1360  4899 ;
                     1360  4900 ; Convert virtual buffer address to physical PTE address
```

```
                                                I 6
                        1360  4901 ;
                  09    EF  1360  4902          EXTZV   S^#VA$V_VPN,-                   ; Get PCBB page number
      51  56    15        1362  4903                  S^#VA$S_VPN,R6,R1                 ;
50    00000000'GF   D0    1365  4904          MOVL    G^MMG$GL_SPTBASE,R0               ; Get SPT address
   78 A5   6041   DE      136C  4905          MOVAL   (R0)[R1],UCB$L_SVAPTE(R5)         ; Set PTE address
    00000000'GF   16      1371  4906          JSB     G^IOC$LOADUBAMAP                  ; Load the PCBB map registers
        02  07    EF      1377  4907          EXTZV   #7,#2,-                           ; Get BA16-BA17
      51  34 A4           137A  4908                  CRB$L_INTD+VEC$W_MAPREG(R4),R1    ;
   51   51  10    78      137D  4909          ASHL    #16,R1,R1                         ; Move to high word
     51  7C A5    B0      1381  4910          MOVW    UCB$W_BOFF(R5),R1                 ; Set BA0-BA8
        34 A4    F0       1385  4911          INSV    CRB$L_INTD+VEC$W_MAPREG(R4),-     ; Set BA9-BA15
   51   07  09            1388  4912                  #9,#7,R1                          ;
     40 A3   51    D0      138B  4913          MOVL    R1,IRP$L_XE_DGUNI(R3)             ; Save UNIBUS address
     54   10 A4   D0      138F  4914          MOVL    CRB$L_AUXSTRUC(R4),R4             ; Get CDB address
            56 8ED0         1393  4915          POPL    R6                                ; Restore register
                            1396  4916          SETIPL  UCB$B_DIPL(R5)                    ; Sync access to device
   0088 D4   63    0E       139A  4917          INSQUE  (R3),@CDB_Q_INPUT+4(R4)           ; Insert at END of queue
          011A   31         139F  4918          BRW     LOAD_PORT
                            13A2  4919 ;
                            13A2  4920 ; Insufficient map registers
                            13A2  4921 ;
            56 8ED0         13A2  4922 20$:     POPL    R6                                ; Restore R6
   50   0344 8F   3C        13A5  4923          MOVZWL  #SS$_INSFMAPREG,R0                ; Else, set error return
          0B9E   31         13AA  4924          BRW     IO_DONE                           ; Finish the request
                            13AD  4925          .DSABL  LSB
```

```
                        13AD  4927          .SBTTL  FILLRCVLIST - FILL RECEIVE BUFFER LIST
                        13AD  4928 ;++
                        13AD  4929 ; FILLRCVLIST - FILL RECEIVE BUFFER LIST
                        13AD  4930 ;
                        13AD  4931 ; Functional description:
                        13AD  4932 ;
                        13AD  4933 ; This routine fills the receive buffer list up to the quota allocated
                        13AD  4934 ; at unit initialization.  It also gives any receive buffers allocated
                        13AD  4935 ; to the receiver.
                        13AD  4936 ;
                        13AD  4937 ; Inputs:
                        13AD  4938 ;
                        13AD  4939 ;       R2 = Buffer Address (ADDRCVLIST ONLY)
                        13AD  4940 ;       R4 = CDB address
                        13AD  4941 ;
                        13AD  4942 ;       IPL = FIPL
                        13AD  4943 ;
                        13AD  4944 ; Outputs:
                        13AD  4945 ;
                        13AD  4946 ;       R0-R2 is destroyed.
                        13AD  4947 ;       All other registers are preserved.
                        13AD  4948 ;
                        13AD  4949 ;--
                        13AD  4950          .ENABL  LSB
                        13AD  4951 FILLRCVLIST::                        ; Fill receive buffer list
                52   D4  13AD  4952          CLRL    R2                  ; No buffer here
                        13AF  4953
                        13AF  4954 ADDRCVLIST::                         ; Add a buffer to the receive list
                38   BB  13AF  4955          PUSHR   #^M<R3,R4,R5>       ; Save registers
          55  00C8 C4   D0  13B1  4956          MOVL    CDB_L_UCB0(R4),R5   ; Get UCB address of unit #0
                01   E1  13B6  4957          BBC     #CDB_STS_V_RUN,-    ; If BC device not running
            5D 025A C4      13B8  4958                  CDB_B_STS(R4),40$
                        13BC  4959
                00B8 C4   B1  13BC  4960 10$:   CMPW    CDB_W_BSZ(R4),-     ; Can new block be allocated ?
                00BA C4       13C0  4961                CDB_W_QUOTA(R4)
                      3B  1A  13C3  4962          BGTRU   35$                 ; If GTRU then no, stop loop
                      52  D5  13C5  4963          TSTL    R2                  ; Buffer already allocated?
                      1B  12  13C7  4964          BNEQ    20$                 ; Br if so
                        13C9  4965
                      51  D4  13C9  4966          CLRL    R1                  ; Zero size
                          A1  13CB  4967          ADDW3   #CXB$C_HEADER+-     ; Determine block size needed
                        13CC  4968                  CXB$C_TRAILER,-         ;
          51  00B8 C4   004C 8F   13CC  4969                  CDB_W_BSZ(R4),R1    ;
                00000000'GF  16  13D3  4970          JSB     G^EXE$ALONONPAGED   ; Allocate the memory
                      1C  50  E9  13D9  4971          BLBC    R0,30$              ; If failure then done
                08 A2     51  B0  13DC  4972          MOVW    R1,CXB$W_SIZE(R2)   ; Insert size
                          1B  90  13E0  4973          MOVB    S^#DYN$C_CXB,-      ; Insert type
                       0A A2      13E2  4974                  CXB$B_TYPE(R2)      ;
                        13E4  4975
                00B8 C4   A2  13E4  4976 20$:   SUBW    CDB_W_BSZ(R4),-     ; Subtract from quota
                00BA C4       13E8  4977                CDB_W_QUOTA(R4)     ;
                20 A2     08  90  13EB  4978 25$:   MOVB    #XE_FC_V_RECV,CXB$B_XE_FUNC(R2) ; Set function request
              009C C4     62  0E  13EF  4979          INSQUE  (R2),CDB_Q_RCVBUF(R4)   ; Insert block on list
                      52  D4  13F4  4980          CLRL    R2                  ; No more buffers given
                      C4  11  13F6  4981          BRB     10$                 ; Continue
                        13F8  4982 ;
                        13F8  4983 ; Buffer allocation failure
```

```
                     13F8  4984 ;
                     13F8  4985 30$:     SETBIT  #XM$V_STS_BUFFAIL,-     ; Set buffer alloc failure
                     13F8  4986                  CDB_L_DEVDEPEND(R4)     ;
           24   11   13FE  4987          BRB     50$                     ; And give any receives to device
                     1400  4988
                     1400  4989 35$:     CLRBIT  #XM$V_STS_BUFFAIL,-     ; Clear buffer alloc failure
                     1400  4990                  CDB_L_DEVDEPEND(R4)     ;
           52   D5  1406  4991          TSTL    R2                      ; Any buffer?
           1A   13  1408  4992          BEQL    50$                     ; Br if not
 02B8      C4   91  140A  4993          CMPB    CDB_B_AQUOTA(R4),-      ; Can we use the additional quota?
 02B9      C4       140E  4994                  CDB_B_MQUOTA(R4)        ;
           06   1E  1411  4995          BGEQU   40$                     ; Br if not
 02B8      C4   96  1413  4996          INCB    CDB_B_AQUOTA(R4)        ; Else, increment the additional
           D2   11  1417  4997          BRB     25$                     ; Use buffer, but don't let
                     1419  4998                                          ;  QUOTA go negative
                     1419  4999
      50   52   D0  1419  5000 40$:     MOVL    R2,R0                   ; Get address of buffer
           06   13  141C  5001          BEQL    50$                     ; Br if none
 00000000'GF    16  141E  5002          JSB     G^COM$DRVDEALMEM        ; Deallocate buffer
                     1424  5003
           03   10  1424  5004 50$:     BSBB    START_RECEIVE           ; Start the receives
           38   BA  1426  5005          POPR    #^M<R3,R4,R5>           ; Restore registers
                05  1428  5006          RSB
                     1429  5007          .DSABL  LSB
```

```
                                1429  5009          .SBTTL  START_RECEIVE - START ANY RECEIVE REQUESTS PENDING
                                1429  5010 ;++
                                1429  5011 ; START_RECEIVE - START ANY RECEIVE REQUESTS
                                1429  5012 ;
                                1429  5013 ; Functional description:
                                1429  5014 ;
                                1429  5015 ; This routine attemptes to start any receives that may be pending.  This
                                1429  5016 ; involves dequeueing a free receive buffer, mapping it, and loading it's
                                1429  5017 ; address and size into the device.
                                1429  5018 ;
                                1429  5019 ; Inputs:
                                1429  5020 ;
                                1429  5021 ;       R4 = CDB address
                                1429  5022 ;       R5 = UCB address of unit # 0
                                1429  5023 ;
                                1429  5024 ;       IPL = FIPL
                                1429  5025 ;
                                1429  5026 ; Outputs:
                                1429  5027 ;
                                1429  5028 ;       R0-R3 are destroyed.
                                1429  5029 ;       All other registers are preserved.
                                1429  5030 ;
                                1429  5031 ;--
                                1429  5032
                                1429  5033 START_RECEIVE::                          ; Start receive operation
                                1429  5034          PUSHQ   R6                      ; Save R6, R7
                          01    E1    142C  5035          BBC     #CDB_STS_V_RUN,-        ; Br if device is not running
                       0F 025A  C4          142E  5036                  CDB_B_STS(R4),10$       ;
                57   1A A4    08    00    EB    1432  5037 5$:       FFC     #0,#MAX_C_RCV,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
                          07    13          1438  5038          BEQL    10$             ; Br if none - just exit
                   53   009C  D4    0F    143A  5039          REMQUE  @CDB_Q_RCVBUF(R4),R3    ; Get a free buffer
                          04    1C          143F  5040          BVC     20$             ; Br if buffer found
                                1441  5041 10$:     POPQ    R6                      ; Restore R6, R7
                          05    1444  5042          RSB                             ; Return to caller
                                1445  5043 ;
                                1445  5044 ; Mark slot in use and create buffer address/character count image
                                1445  5045 ; in receive buffer and load UNIBUS adapter map registers.
                                1445  5046 ;
                                1445  5047 20$:     SETBIT  R7,CDB_B_RCVMAP(R4)     ; Mark slot in use
             22 A3    57    90    144A  5048          MOVB    R7,CXB$B_XE_SLOT(R3)    ; Save mapping slot index
             52    24 A5    D0    144E  5049          MOVL    UCB$L_CRB(R5),R2        ; Get CRB address
                                1452  5050
                                1452  5051          .IF DF  RCV_DATAP
                                1452  5052 ;
                                1452  5053 ; The following instruction also sets the default data path to the
                                1452  5054 ; Direct Data Path.
                                1452  5055 ;
                                1452  5056          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                                1452  5057          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
                                1452  5058          MOVL    CDB_L_RCVMAP(R4)[R7],-  ; Store mapping information
                                1452  5059                   CRB$L_INTD+VEC$W_MAPREG(R2) ; into CRB.
                                1452  5060 ;
                                1452  5061 ; Try to allocate a buffered data path. If successful then use it, else
                                1452  5062 ; use the Direct Data Path.
                                1452  5063 ;
                                1452  5064 ;*** The driver no longer uses the Buffered Data Paths for receive operations
                                1452  5065 ;*** because there is no upper limit to the amount of time that the driver
```

```
                        1452  5066 ;*** can hang onto the data path. Therefore, if there are other system devices
                        1452  5067 ;*** (such as RK07s) which need the data paths then there is no guarantee as to
                        1452  5068 ;*** when they will get one.
                        1452  5069 ;***
                        1452  5070          PUSHL   R2                              ; Save R2
                        1452  5071          JSB     G^IOC$REQDATAPNW                ; Try to get a buffered data path
                        1452  5072          POPL    R2                              ; Restore R2
                        1452  5073          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                        1452  5074          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
                        1452  5075          MOVL    CRB$L_INTD+VEC$W_MAPREG[R2],-   ; Save number of data path
                        1452  5076                   CDB_L_RCVMAP(R4)[R7]           ;  used.
                        1452  5077
                        1452  5078          .ENDC   ; DF RCV_DATAP
                        1452  5079 ;
                        1452  5080 ; Find next ring entry and insert data
                        1452  5081 ;
      56    19 A4   9A  1452  5082          MOVZBL  CDB_B_NEXTRCV(R4),R6    ; Get next ring entry
            19 A4   96  1456  5083          INCB    CDB_B_NEXTRCV(R4)       ; Bump ring pointer
            F8 8F   8A  1459  5084          BICB    #^C<MAX_C_RCV-1>,-      ; Modulo rcv ring size
            19 A4       145C  5085                   CDB_B_NEXTRCV(R4)      ;
      23 A3    56   90  145E  5086          MOVB    R6,CXB$B_XE_RING(R3)    ; Save ring entry number
   56    5C A446   D0  1462  5087          MOVL    CDB_L_RRADDR(R4)[R6],R6 ; Retrieve next ring entry address
                        1467  5088          PUSHQ   R3                      ; Save IRP, CDB address
                        146A  5089
                        146A  5090 ;
                        146A  5091 ; Setup the ring descriptor entry for the recieve buffer. The length of the
                        146A  5092 ; buffer, which is a constant, was initiated when the controller was first
                        146A  5093 ; started up, and so is not setup here.
                        146A  5094 ;
                        146A  5095
      51    38 A3   9E  146A  5096          MOVAB   CXB$T_R_DATA(R3),R1     ; Get receive buffer data addr
         02 A6    51   B0  146E  5097          MOVW    R1,RCV_W_SEGBL(R6)      ; Set BA0-BA8
      57    1C A447   DE  1472  5098          MOVAL   CDB_L_RCVMAP(R4)[R7],R7 ; Get mapping info slot address
02 A6 07    09    67   F0  1477  5099          INSV    (R7),#9,#7,RCV_W_SEGBL(R6) ; Set BA9-BA15 from map reg
   50 67    02    07   EF  147D  5100          EXTZV   #7,#2,(R7),R0           ; Get BA16-BA17 also
         04 A6    50   90  1482  5101          MOVB    R0,RCV_B_SEGBH(R6)      ; Set BA16-BA17
      08 A6    22 A3   B0  1486  5102          MOVW    CXB$W_XE_RID(R3),RCV_W_RID(R6) ; Store unique request ID
                        148B  5103
                        148B  5104          .IF DF  RCV_DATAP
                        148B  5105          MOVZBL  CRB$L_INTD+VEC$B_DATAPATH(R2),R4 ; Get data path number
                        148B  5106          .IFF    ; DF RCV_DATAP
               54   D4  148B  5107          CLRL    R4                      ; Use direct data path for rcvs
                        148D  5108          .ENDC
                        148D  5109          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
      52    02 A7   9A  148D  5110          MOVZBL  2(R7),R2                ; Set number of map registers
         53    67   3C  1491  5111          MOVZWL  (R7),R3                 ; Set first map register number
   00000000'GF   16  1494  5112          JSB     G^IOC$LOADUBAMAPN       ; Load the map registers
                        149A  5113          POPQ    R3                      ; Restore IRP, CDB address
                        149D  5114 ;
                        149D  5115 ; Disable interrupts and queue request to input queue
                        149D  5116 ;
                        149D  5117          DSBINT  UCB$B_DIPL(R5)          ; Disable interrupts
      05 A6    80 8F   90  14A4  5118          MOVB    #RCV_FLG_M_OWN,RCV_B_FLAGS(R6) ; Give buffer to DEUNA/DELUA
            00B6 C4   96  14A9  5119          INCB    CDB_B_RCVCNT(R4)        ; Tally one more receive in progress
      0088 D4   63   0E  14AD  5120          INSQUE  (R3),@CDB_Q_INPUT+4(R4) ; Insert receive buffer on
                        14B2  5121                                          ;   input queue
               02   12  14B2  5122          BNEQ    30$                     ; Br if not first entry on queue
```

```
                06   10  14B4  5123            BSBB    LOAD_PORT               ; Request port % give request to DEUNA/DELUA
                         14B6  5124 30$:        ENBINT                          ; Re-enable interrupts
              FF76   31  14B9  5125            BRW     5$                      ; Let's try it again
```

```
                                14BC  5127          .SBTTL  LOAD_PORT - LOAD CSR'S WITH COMMAND REQUEST
                                14BC  5128 ;++
                                14BC  5129 ; LOAD_PORT - LOAD CSR'S WITH COMMAND REQUEST
                                14BC  5130 ;
                                14BC  5131 ; Functional description:
                                14BC  5132 ;
                                14BC  5133 ; This routine loads the CSR's and PCBB with a command to process.
                                14BC  5134 ;
                                14BC  5135 ; Inputs:
                                14BC  5136 ;
                                14BC  5137 ;       R4 = CDB address
                                14BC  5138 ;       R5 = UCB address
                                14BC  5139 ;
                                14BC  5140 ;       IPL = DIPL
                                14BC  5141 ;
                                14BC  5142 ; Outputs:
                                14BC  5143 ;
                                14BC  5144 ;       R4,R5 are preserved.
                                14BC  5145 ;
                                14BC  5146 ;       R0-R3 may be destroyed.
                                14BC  5147 ;
                                14BC  5148 ;--
                                14BC  5149 LOAD_PORT::                                  ; Load port command
      14 025A C4    03    E5    14BC  5150          BBCC    #CDB_STS_V_FUNC,CDB_B_STS(R4),10$ ; Br if function pending
           52   24 A5    D0    14C2  5151          MOVL    UCB$L_CRB(R5),R2            ; Get CRB address
                                14C6  5152          ASSUME  IDB$L_CSR EQ 0
           52   2C B2    D0    14C6  5153          MOVL    @CRB$L_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
      53   0084 D4    0F    14CA  5154          REMQUE  @CDB_Q_INPUT(R4),R3         ; Get next IRP
                   06    1C    14CF  5155          BVC     20$                         ; Br if got one
      025A C4    08    88    14D1  5156          BISB2   #CDB_STS_M_FUNC,CDB_B_STS(R4) ; Indicate no function pending
                   05    14D6  5157 10$:    RSB                                     ; Return to caller
                         14D7  5158
      0223 C4    07    90    14D7  5159 20$:    MOVB    #DNI_TIM,CDB_B_TIM_DNI(R4) ; Start timer on DNI bit
        1B   0A A3    91    14DC  5160          CMPB    IRP$B_TYPE(R3),#DYN$C_CXB ; Is this a CXB?
                   0F    13    14E0  5161          BEQL    30$                         ; Br if yes
      51   00D6 C4    9E    14E2  5162          MOVAB   CDB_G_PCBB(R4),R1           ; Get PCBB address
        0A   0A A3    91    14E7  5163          CMPB    IRP$B_TYPE(R3),#DYN$C_IRP ; Is this an IRP?
                   2A    13    14EB  5164          BEQL    90$                         ; Br if yes
                         14ED  5165          BUG_CHECK NOBUFPCKT,FATAL               ; Fatal error - what is it???
                         14F1  5166 ;
                         14F1  5167 ; Dispatch of CXB request
                         14F1  5168 ;
                         14F1  5169 30$:    $DISPATCH       CXB$B_XE_FUNC(R3),TYPE=B,- ; Dispatch on function request
                         14F1  5170          <-      ;function       action
                         14F1  5171                  <XE_FC_V_XMIT   40$>,-  ; XMIT requested
                         14F1  5172                  <XE_FC_V_RECV   50$>,-  ; RECV requested
                         14F1  5173          >
                         14FA  5174
                         14FA  5175          BUG_CHECK NOBUFPCKT,FATAL               ; Fatal error - not a valid IRP
                         14FE  5176
                         14FE  5177 ;
                         14FE  5178 ; XMIT request
                         14FE  5179 ;
      0098 D4    63    0E    14FE  5180 40$:    INSQUE  (R3),@CDB_Q_XMTPND+4(R4) ; Insert CXB on WAITING queue
      0222 C4    05    90    1503  5181          MOVB    #XMT_TIM,CDB_B_TIM_XMT(R4) ; Start xmit timer
              48 8F    90    1508  5182          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_PDMD,- ; Load command
                   62          150B  5183                  PCSR0(R2)                   ;  and enable interrupts
```

```
                     05  150C  5184          RSB                                 ; Return to caller
                         150D  5185 ;
                         150D  5186 ; RECV request
                         150D  5187 ;
                 00A8 D4    63  150D  5188 50$:     INSQUE  (R3),@CDB_Q_RCVPND+4(R4) ; Insert CXB on WAITING queue
                       48 8F    90  1512  5189          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_PDMD,- ; Load command
                            62        1515  5190                  PCSR0(R2)                   ;  and enable interrupts
                     05  1516  5191          RSB                                 ; Return to caller
                         1517  5192
                         1517  5193 ;
                         1517  5194 ; Dispatch of IRP request
                         1517  5195 ;
                         1517  5196 90$:     $DISPATCH       IRP$B_XE_FUNC(R3),TYPE=B,- ; Dispatch on function request
                         1517  5197          <-      ;function       action
                         1517  5198
                         1517  5199                  <XE_FC_V_INIT   IN_INIT>,-      ; Initialize request
                         1517  5200                  <XE_FC_V_RESTART IN_INIT>,-     ; Automatic restart
                         1517  5201                  <XE_FC_V_TESTD  IN_TESTD>,-     ; Self-test done
                         1517  5202                  <XE_FC_V_LDRING IN_RING>,-      ; Setup ring descriptor
                         1517  5203                  <XE_FC_V_RDDPA  IN_RDDPA>,-     ; Read default physical address
                         1517  5204                  <XE_FC_V_WTPHYAD IN_PHYAD>,-    ; Write physical address
                         1517  5205                  <XE_FC_V_WTMULTI IN_MULTI>,-    ; Write multicast address list
                         1517  5206                  <XE_FC_V_WTMODE IN_MODE>,-      ; Define mode
                         1517  5207                  <XE_FC_V_START  IN_START>,-     ; Start XMIT/RECV process
                         1517  5208                  <XE_FC_V_RDCNTS IN_RDCNTS>,-    ; Read counters
                         1517  5209                  <XE_FC_V_RCCNTS IN_RCCNTS>,-    ; Read and clear counters
                         1517  5210                  <XE_FC_V_MULTI  IN_MULTIC>,-    ; Write new multicast list
                         1517  5211                  <XE_FC_V_LDMEM  IN_LDMEM>,-     ; Load DEUNA/DELUA memory
                         1517  5212                  <XE_FC_V_RDMEM  IN_RDMEM>,-     ; Read DEUNA/DELUA memory
                         1517  5213                  <XE_FC_V_UPST   IN_UPST>,-      ; Load and start micro-address
                         1517  5214                  <XE_FC_V_LDSID  IN_LDSID>,-     ; Load system ID parameters
                         1517  5215                  <XE_FC_V_RDSID  IN_RDSID>,-     ; Read system ID parameters
                         1517  5216                  <XE_FC_V_STOPPR IN_STOPPR>,-    ; Stop xmit and receive process
                         1517  5217                  <XE_FC_V_STEST  IN_STEST>,-     ; Perform DEUNA/DELUA self test
                         1517  5218                  <XE_FC_V_READST IN_READST>,-    ; Read port status
                         1517  5219                  <XE_FC_V_CHMODE IN_CHMODE>,-    ; Change the mode definition
                         1517  5220                  <XE_FC_V_STOP   IN_RSMODE>,-    ; Reset the mode definition
                         1517  5221                  <XE_FC_V_MOPCTR IN_RDCNTS>,-    ; MOP read counters request
                         1517  5222                  <XE_FC_V_ESTEST IN_ESTEST>,-    ; Perform DELUA extended self test
                         1517  5223                  <XE_FC_V_HALT   IN_HALT>,-      ; Halt port (DELUA only)
                         1517  5224          >
                         1554  5225
                         1554  5226          BUG_CHECK NOBUFPCKT,FATAL           ; Fatal error - not a valid IRP
                         1558  5227
                         1558  5228
                         1558  5229 IN_INIT:                                      ; Initialize DEUNA/DELUA request
                         1558  5230 ;
                         1558  5231 ;  Perform SELF-TEST on the UNA
                         1558  5232 ;
                      43 0F    90  1558  5233          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_STEST,- ; Load self-test command
                         62        155B  5234                  PCSR0(R2)                   ;  and enable interrupts
                 21 A3    01    90  155C  5235          MOVB    #XE_FC_V_TESTD,IRP$B_XE_FUNC(R3) ; Set up next function
                       00E6    31  1560  5236          BRW     CONTIN                      ; Contine the request
                         1563  5237 ;
                         1563  5238 ; Check completion status of self-test and load PCBB address if okay
                         1563  5239 ;
                         1563  5240 IN_TESTD:                                     ; SELF-TEST done
```

```
                0070 8F     AB  1563  5241          BICW3   #XE_PCSR1_M_DEVID,-      ; Get DEUA/DELUA status,
          50      02 A2         1567  5242                  PCSR1(R2),R0             ;  minus device id
          50        02    B1    156A  5243          CMPW    #XE_STAT_V_READY,R0      ; Is the DEUNA/DELUA okay?
                    1E    13    156D  5244          BEQL    20$                      ; Br if YES - good DEUNA/DELUA
    00C4 C4      03 A2    90    156F  5245          MOVB    PCSR1+1(R2),CDB_L_DEVDEPEND(R4) ; Save self-test status code
       50      0054 8F    3C    1575  5246          MOVZWL  #SS$_CTRLERR,R0          ; Assume broken DEUNA/DELUA
     0B 02 A2     0F    E1    157A  5247          BBC     #XE_PCSR1_V_XPWR,PCSR1(R2),10$ ; Br if the XCVR power is okay
       50      01A4 8F    3C    157F  5248          MOVZWL  #SS$_MEDOFL,R0           ; Else, return the correct indicator
                         1584  5249          SETBIT  #XM$D_STS_DISC,CDB_L_DEVDEPEND(R4) ; Save Internal error reason
                  0262    31    158A  5250 10$:     BRW     REQ_COM_ERR              ; Complete request in error
                         158D  5251
    04 A2      00C0 C4    B0    158D  5252 20$:     MOVW    CDB_L_PCBBUA(R4),PCBB0(R2) ; Load PCBB UNIBUS address
    06 A2      00C2 C4    90    1593  5253          MOVB    CDB_L_PCBBUA+2(R4),PCBB2(R2) ;
                  41 8F    90    1599  5254          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETPCBB,- ; Load command
                     62          159C  5255                  PCSR0(R2)                ;  and enable interrupts
             21 A3    02    90    159D  5256          MOVB    #XE_FC_V_LDRING,IRP$B_XE_FUNC(R3) ; Set next function
                  00A5    31    15A1  5257          BRW     CONTIN                   ; Contine the request
                         15A4  5258
                         15A4  5259 IN_RING:                                      ; Setup ring descriptor
                     61    09    B0    15A4  5260          MOVW    #PB_FC_V_WTRING,PBB_B_FUNC(R1) ; Set function request
          50      00C0 C4    D0    15A7  5261          MOVL    CDB_L_PCBBUA(R4),R0      ; Get PCBB UNIBUS address
          50        08    C0    15AC  5262          ADDL    #CDB_G_XRUDB-CDB_G_PCBB,R0 ; Compute RING DESCRIPTOR address
             02 A1     50    D0    15AF  5263          MOVL    R0,PBB_W_PCBB2(R1)       ; Store RING DESCRIPTOR address
                  42 8F    90    15B3  5264          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
                     62          15B6  5265                  PCSR0(R2)                ;  and enable interrupts
             21 A3    15    90    15B7  5266          MOVB    #XE_FC_V_RDDPA,IRP$B_XE_FUNC(R3) ; Set next function
                  008B    31    15BB  5267          BRW     CONTIN                   ; Continue request
                         15BE  5268
                         15BE  5269 IN_RDDPA:
                     61    02    B0    15BE  5270          MOVW    #PB_FC_V_RDDEFAULT,PBB_B_FUNC(R1) ; Set function request
                  42 8F    90    15C1  5271          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
                     62          15C4  5272                  PCSR0(R2)                ;  and enable interrupts
             21 A3    03    90    15C5  5273          MOVB    #XE_FC_V_WTPHYAD,IRP$B_XE_FUNC(R3) ; Set next function
                     7E    11    15C9  5274          BRB     CONTIN                   ; Continue request
                         15CB  5275
                         15CB  5276 IN_PHYAD:                                     ; Write physical address
       0269 C4      02 A1    D0    15CB  5277          MOVL    PBB_W_PCBB2(R1),CDB_G_HWA(R4) ; Save hardware physical address
       026D C4      06 A1    B0    15D1  5278          MOVW    PBB_W_PCBB6(R1),CDB_G_HWA+4(R4) ;       ...
       026F C4      02 A1    D0    15D7  5279          MOVL    PBB_W_PCBB2(R1),CDB_G_PHYADR(R4) ; Save current address
       0273 C4      06 A1    B0    15DD  5280          MOVW    PBB_W_PCBB6(R1),CDB_G_PHYADR+4(R4) ;    ...
             27 0263 C4    E8    15E3  5281          BLBS    CDB_G_PHA(R4),IN_MULTI   ; Br if physical address not there
                     61    05    B0    15E8  5282          MOVW    #PB_FC_V_WTPHYAD,PBB_B_FUNC(R1) ; Set function request
          02 A1      0263 C4    D0    15EB  5283          MOVL    CDB_G_PHA(R4),PBB_W_PCBB2(R1) ; Store physical address
          06 A1      0267 C4    B0    15F1  5284          MOVW    CDB_G_PHA+4(R4),PBB_W_PCBB6(R1) ;       ...
    026F C4      0263 C4    D0    15F7  5285          MOVL    CDB_G_PHA(R4),CDB_G_PHYADR(R4) ; Store current address
    0273 C4      0267 C4    B0    15FE  5286          MOVW    CDB_G_PHA+4(R4),CDB_G_PHYADR+4(R4) ;    ...
                  42 8F    90    1605  5287          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
                     62          1608  5288                  PCSR0(R2)                ;  and enable interrupts
             21 A3    04    90    1609  5289          MOVB    #XE_FC_V_WTMULTI,IRP$B_XE_FUNC(R3) ; Set next function
                     3A    11    160D  5290          BRB     CONTIN                   ; Continue request
                         160F  5291
                         160F  5292 IN_MULTI:                                     ; Write multicast address list
                0276 C4    95    160F  5293          TSTB    CDB_B_MULTI(R4)          ; Any entries?
                     23    13    1613  5294          BEQL    IN_MODE                  ; Br if none ..do a write mode
                     61    07    B0    1615  5295          MOVW    #PB_FC_V_WTMULTI,PBB_B_FUNC(R1) ; Set function request
          50      00C0 C4    D0    1618  5296          MOVL    CDB_L_PCBBUA(R4),R0      ; Get PCBB UNIBUS address
    50    00000054 8F    C0    161D  5297          ADDL    #CDB_G_MUDB-CDB_G_PCBB,R0 ; Compute address of multicast list
```

```
                02 A1    50   D0  1624  5298          MOVL    R0,PBB_W_PCBB2(R1)      ; Store address of multicast list
          05 A1   0276   C4   90  1628  5299          MOVB    CDB_B_MULTI(R4),PBB_W_PCBB4+1(R1) ; Set number of entries
                    42   8F   90  162E  5300          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
                         62       1631  5301                  PCSR0(R2)               ;  and enable interrupts
                21 A3    05   90  1632  5302          MOVB    #XE_FC_V_WTMODE,IRP$B_XE_FUNC(R3) ; Set next function
                         11   11  1636  5303          BRB     CONTIN                  ; Continue request
                              1638  5304
                              1638  5305  IN_MODE:                                   ; Define mode
                         61   0D   90  1638  5306          MOVB    #PB_FC_V_WTMODE,PBB_B_FUNC(R1) ; Set function request
          02 A1    0258   C4   B0  163B  5307          MOVW    CDB_W_MODE(R4),PBB_W_PCBB2(R1) ; Define new mode
                    42   8F   90  1641  5308          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
                         62       1644  5309                  PCSR0(R2)               ;  and enable interrupts
                21 A3    06   90  1645  5310          MOVB    #XE_FC_V_START,IRP$B_XE_FUNC(R3) ; Set next function
             0084 C4     63   0E  1649  5311  CONTIN: INSQUE  (R3),CDB_Q_INPUT(R4)     ; Insert at front of input queue
                              05  164E  5312          RSB
                              164F  5313
                              164F  5314  IN_CHMODE:                                 ; Change the mode definition
                              164F  5315          ASSUME  NMA$C_STATE_ON EQ 0
                              164F  5316          ASSUME  NMA$C_STATE_OFF EQ 1
                         50   D4  164F  5317          CLRL    R0                      ; Assume we don't have to define mode
                   0C 00DB   C5   E8  1651  5318          BLBS    UCB$B_XE_PRM(R5),10$    ; Br if not promiscuous user
                         0F   E2  1656  5319          BBSS    #PB_MOD_V_PROM,-        ; Else change to promiscuous mode
                   06 0258   C4       1658  5320                  CDB_W_MODE(R4),10$      ; ..and skip if already set
                      025D   C4   94  165C  5321          CLRB    CDB_B_PRM(R4)           ; Set new characteristic in CDB
                         50   D6  1660  5322          INCL    R0                      ; Indicate we have to define the mode
                   0D 00DC   C5   E8  1662  5323  10$:     BLBS    UCB$B_XE_MLT(R5),20$    ; Br if all multicasts not enabled
                         0E   E2  1667  5324          BBSS    #PB_MOD_V_ENAL,-        ; Else, set to enable all mlts
                   07 0258   C4       1669  5325                  CDB_W_MODE(R4),20$      ; ..and skip if already enabled
                      025E   C4   94  166D  5326          CLRB    CDB_B_MLT(R4)           ; Set new characteristic in CDB
                      50     01   C8  1671  5327          BISL    #1,R0                   ; Indicate need to define the mode
                         42   50   E9  1674  5328  20$:     BLBC    R0,IN_MULTIC            ; Br if we don't have to define mode
                         61   0D   90  1677  5329          MOVB    #PB_FC_V_WTMODE,PBB_B_FUNC(R1) ; Set function request
          02 A1    0258   C4   B0  167A  5330          MOVW    CDB_W_MODE(R4),PBB_W_PCBB2(R1) ; Define new mode
                    42   8F   90  1680  5331          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
                         62       1683  5332                  PCSR0(R2)               ;  and enable interrupts
                21 A3    0C   90  1684  5333          MOVB    #XE_FC_V_MULTI,IRP$B_XE_FUNC(R3) ; Set next function
                         BF   11  1688  5334          BRB     CONTIN                  ; Wait for next function
                              168A  5335
                              168A  5336  IN_RSMODE:                                ; Reset the mode definition
                         61   0D   90  168A  5337          MOVB    #PB_FC_V_WTMODE,PBB_B_FUNC(R1) ; Set function request
          02 A1    0258   C4   B0  168D  5338          MOVW    CDB_W_MODE(R4),PBB_W_PCBB2(R1) ; Define new mode
                         40   11  1693  5339          BRB     DONE                    ; Request is complete
                              1695  5340
                              1695  5341                  .ENABL  LSB
                              1695  5342  IN_RCCNTS:                                ; Read and clear counters request
                         61   0B   90  1695  5343          MOVB    #PB_FC_V_RCCNTS,PBB_B_FUNC(R1) ; Set function request
                         03   11  1698  5344          BRB     10$
                              169A  5345  IN_RDCNTS:                                ; Read counters request
                         61   0A   90  169A  5346          MOVB    #PB_FC_V_RDCNTS,PBB_B_FUNC(R1) ; Set function request
                50    00C0   C4   D0  169D  5347  10$:     MOVL    CDB_L_PCBBUA(R4),R0     ; Get PCBB UNIBUS address
                      50     14   C0  16A2  5348          ADDL    #CDB_G_CUDB-CDB_G_PCBB,R0 ; Compute Counter UDB address
                   02 A1     50   D0  16A5  5349          MOVL    R0,PBB_W_PCBB2(R1)      ; Store Counter UDB address
          06 A1    0040   8F   B0  16A9  5350          MOVW    #UDB_CTR_K_LENGTH,PBB_W_PCBB6(R1) ; Set size of counter blk
                    42   8F   90  16AF  5351  PEND1:   MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load port command
                         62       16B2  5352                  PCSR0(R2)               ;  and enable interrupts
             0090 D4     63   0E  16B3  5353  PEND:    INSQUE  (R3),@CDB_Q_DONE+4(R4)  ; Post on waiting queue
                              05  16B8  5354          RSB
```

```
                        16B9  53 5          .DSABL  LSB
                        16B9  5356
                        16B9  5357 IN_MULTIC:                           ; Write new multicast address list
             1483   30  16B9  5358          BSBW    MOVE_MULTI             ; Copy the multicast address list
       61    07   B0  16BC  5359          MOVW    #PB_FC_V_WTMULTI,PBB_B_FUNC(R1) ; Set function request
   50  00C0 C4   D0  16BF  5360          MOVL    CDB_L_PCBBUA(R4),R0    ; Get PCBB UNIBUS address
50  00000054 8F   C0  16C4  5361          ADDL    #CDB_G_MUDB-CDB_G_PCBB,R0 ; Compute address of multicast list
    02 A1    50   D0  16CB  5362          MOVL    R0,PBB_W_PCBB2(R1)     ; Store address of multicast list
05 A1  0276 C4   90  16CF  5363          MOVB    CDB_B_MULTI(R4),PBB_W_PCBB4+1(R1) ; Set number of entries
         42 8F   90  16D5  5364 DONE:    MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Load command
             62             16D8  5365                  PCSR0(R2)         ;  and enable interrupts
            0110   31  16D9  5366          BRW     REQ_COM                ; Request is complete
                        16DC  5367
                        16DC  5368 IN_LDMEM:                            ; Load DEUNA/DELUA memory
       61    11   B0  16DC  5369          MOVW    #PB_FC_V_LOADMEM,PBB_B_FUNC(R1) ; Set function request
   50  00C0 C4   D0  16DF  5370          MOVL    CDB_L_PCBBUA(R4),R0    ; Get PCBB UNIBUS address
50  00000090 8F   C0  16E4  5371          ADDL    #CDB_G_SCRUDB-CDB_G_PCBB,R0 ; Compute address of scratch UDB
    02 A1    50   D0  16EB  5372          MOVL    R0,PBB_W_PCBB2(R1)     ; Store address of scratch UDB
    50   32 A3   B0  16EF  5373          MOVW    IRP$W_BCNT(R3),R0      ; Get user buffer length
                        16F3  5374 ;
                        16F3  5375 ;       The following r unding is required by the DEUNA/DELUA hardware.
                        16F3  5376 ;       The rounding should have no effect on the buffer size, since
                        16F3  5377 ;       the allocation routine rounds to a multiple of 16 bytes anyway.
                        16F3  5378 ;
                        16F3  5379          ASSUME  P2B_T_DATA LE 12       ; For rounding to be valid
             50   D6   16F3  5380          INCL    R0                     ; Round up buffer size
       50    01   AA  16F5  5381          BICW    #1,R0                  ;       ...
   0166 C4    50   B0  16F8  5382          MOVW    R0,CDB_G_SCRUDB(R4)    ; Set size of load
         40 A3   D0  16FD  5383          MOVL    IRP$L_XE_DGUNI(R3),-   ; Store user UNIBUS buffer address
       0168 C4         1700  5384                  CDB_G_SCRUDB+2(R4)     ;
         44 A3   B0  1703  5385          MOVW    IRP$L_XE_UPADR(R3),-   ; Set IDBB internal address
       016C C4         1706  5386                  CDB_G_SCRUDB+6(R4)     ;
            A4   11  1709  5387          BRB     PEND1                  ; Request is pending
                        170B  5388
                        170B  5389 IN_RDMEM:                            ; Read DEUNA/DELUA memory
       61    10   B0  170B  5390          MOVW    #PB_FC_V_DUMPMEM,PBB_B_FUNC(R1) ; Set function request
   50  00C0 C4   D0  170E  5391          MOVL    CDB_L_PCBBUA(R4),R0    ; Get PCBB UNIBUS address
50  00000090 8F   C0  1713  5392          ADDL    #CDB_G_SCRUDB-CDB_G_PCBB,R0 ; Compute address of scratch UDB
    02 A1    50   D0  171A  5393          MOVL    R0,PBB_W_PCBB2(R1)     ; Store address of scratch UDB
    50   32 A3   B0  171E  5394          MOVW    IRP$W_BCNT(R3),R0      ; Get user buffer length
                        1722  5395 ;
                        1722  5396 ;       The following rounding is required by the DEUNA/DELUA hardware.
                        1722  5397 ;       The rounding should have no effect on the buffer size, since
                        1722  5398 ;       the allocation routine rounds to a multiple of 16 bytes anyway.
                        1722  5399 ;
                        1722  5400          ASSUME  P2B_T_DATA LE 12       ; For rounding to be valid
             50   D6   1722  5401          INCL    R0                     ; Round up buffer size
       50    01   AA  1724  5402          BICW    #1,R0                  ;       ...
   0166 C4    50   B0  1727  5403          MOVW    R0,CDB_G_SCRUDB(R4)    ; Set size of load
         40 A3   D0  172C  5404          MOVL    IRP$L_XE_DGUNI(R3),-   ; Store user UNIBUS buffer address
       0168 C4         172F  5405                  CDB_G_SCRUDB+2(R4)     ;
         44 A3   B0  1732  5406          MOVW    IRP$L_XE_UPADR(R3),-   ; Set IDBB internal address
       016C C4         1735  5407                  CDB_G_SCRUDB+6(R4)     ;
          FF74   31  1738  5408          BRW     PEND1                  ; Request is pending
                        173B  5409
                        173B  5410 IN_UPST:                             ; Load and start micro-address
       61    01   B0  173B  5411          MOVW    #PB_FC_V_LSUADDR,PBB_B_FUNC(R1) ; Set function request
```

```
                44 A3    B0   173E  5412          MOVW    IRP$L_XE_UPADR(R3),-    ; Set IDBB UNIBUS address
                02 A1         1741  5413                  PBB_W_PCBB2(R1)         ;
                 FF8F    31   1743  5414          BRW     DONE                    ; Request is complete
                              1746  5415
                              1746  5416 IN_LDSID:                               ; Load system ID parameters
            61    13    B0    1746  5417          MOVW    #PB_FC_V_WTSYSID,PBB_B_FUNC(R1) ; Set function request
        50  00C0  C4    D0    1749  5418          MOVL    CDB_L_PCBBUA(R4),R0     ; Get PCBB UNIBUS address
50   00000098  8F    C0    174E  5419          ADDL    #CDB_G_SIDUDB-CDB_G_PCBB,R0 ; Compute address of SID UDB
          02 A1    50    D0    1755  5420          MOVL    R0,PBB_W_PCBB2(R1)      ; Store address of SID UDB
                              1759  5421          ASSUME  PBB_L_POINTER EQ 0
            50    2C  B3    D0    1759  5422          MOVL    @IRP$L_SVAPTE(R3),R0    ; Get address of system ID parameters
       016E C4    60    7D    175D  5423          MOVQ    (R0),CDB_G_SIDUDB(R4)   ; Copy system ID parameters
          06 A1    04    B0    1762  5424          MOVW    #4,PBB_W_PCBB6(R1)      ; Write only 4 words - all others
                              1766  5425                                          ; are Read Only!
                 FF6C    31   1766  5426          BRW     DONE                    ; Request is complete
                              1769  5427
                              1769  5428 IN_RDSID:                               ; Read system ID parameters
            61    12    B0    1769  5429          MOVW    #PB_FC_V_RDSYSID,PBB_B_FUNC(R1) ; Set function request
        50  00C0  C4    D0    176C  5430          MOVL    CDB_L_PCBBUA(R4),R0     ; Get PCBB UNIBUS address
50   00000098  8F    C0    1771  5431          ADDL    #CDB_G_SIDUDB-CDB_G_PCBB,R0 ; Compute address of SID UDB
          02 A1    50    D0    1778  5432          MOVL    R0,PBB_W_PCBB2(R1)      ; Store address of SID UDB
                    1E    B0    177C  5433          MOVW    #<UDB_SID_C_LENGTH/2>,- ; Set size of UDB
                06 A1         177E  5434                  PBB_W_PCBB6(R1)         ;
                 FF2C    31   1780  5435          BRW     PEND1                   ; Request is pending
                              1783  5436
                              1783  5437 IN_STEST:                               ; Perform DEUNA/DELUA self-test
                43 8F    90   1783  5438          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_STEST,- ; Load command
                    62         1786  5439                  PCSR0(R2)               ;  and enable interrupts
                    02    8A   1787  5440          BICB    #CDB_STS_M_RUN,-        ; Clear the RUN bit
                025A C4         1789  5441                  CDB_B_STS(R4)           ;
                 FF24    31   178C  5442          BRW     PEND                    ; Hold the request pending
                              178F  5443                                          ;  until DNI is set
                              178F  5444
                              178F  5445 IN_READST:                              ; Read port status
          61    0E    B0    178F  5446          MOVW    #PB_FC_V_RDSTATUS,PBB_B_FUNC(R1) ; Set function request
             FF1A    31   1792  5447          BRW     PEND1                   ; Request is pending
                              1795  5448
                              1795  5449 IN_STOPPR:                              ; Stop XMIT/RECV processes
            4F 8F    90    1795  5450          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_STOP,- ; Load command
                62         1798  5451                  PCSR0(R2)               ;  and enable interrupts
                02    8A   1799  5452          BICB    #CDB_STS_M_RUN,-        ; Clear the RUN bit
          025A C4         179B  5453                  CDB_B_STS(R4)           ;
                4C    11   179E  5454          BRB     REQ_COM                 ; Request is now complete
                              17A0  5455
                              17A0  5456 IN_START:                               ; Start XMIT/RECV processes
            44 8F    90    17A0  5457          MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_START,- ; Load command
                62         17A3  5458                  PCSR0(R2)               ;  and enable interrupts
                02    88   17A4  5459          BISB    #CDB_STS_M_RUN,-        ; Set the RUN bit
          025A C4         17A6  5460                  CDB_B_STS(R4)           ;
       55    1C  A3    D0    17A9  5461          MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
                10    A8   17AD  5462          BISW    #UCB$M_XE_RUN,-         ; Here also
             68 A5         17AF  5463                  UCB$W_DEVSTS(R5)        ;
                03    AA   17B1  5464          BICW    #UCB$M_INT!UCB$M_TIM,-  ; No timeout is possible now
             64 A5         17B3  5465                  UCB$W_STS(R5)           ;
                              17B5  5466          SAVIPL                          ; Save the current IPL for WFIKPCH
                              17B8  5467          WFIKPCH START_CONT,#2           ; Wait 2 seconds for hardware state
                              17C2  5468                                          ;   to get set
```

```
                     17C2  5469 START_CONT:                                ; Continue with startup
                     17C2  5470         CLRBIT  #UCB$V_TIMOUT,UCB$W_STS(R5) ; Clear timeout status
52   24 A5    D0     17C7  5471         MOVL    UCB$L_CRB(R5),R2          ; Get CRB address
                     17CB  5472         ASSUME  IDB$L_CSR EQ 0
52   2C B2    D0     17CB  5473         MOVL    @CRB$L_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
     55       DD     17CF  5474         PUSHL   R5                        ; Save UCB address
     19       10     17D1  5475         BSBB    REQ_COM                   ; Finish the startup request
     55    8ED0      17D3  5476         POPL    R5                        ; Restore R5
           05        17D6  5477         RSB                               ; Return to timout processing
                     17D7  5478
                     17D7  5479 IN_ESTEST:                                ; Perform the DELUA extended self-test
   4B 8F     90      17D7  5480         MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_ESTEST,- ; Load command
     62               17DA  5481                 PCSR0(R2)                 ;  and enable interrupts
     02     8A        17DB  5482         BICB    #CDB_STS_M_RUN,-          ; Clear the RUN bit
 025A C4               17DD  5483                 CDB_B_STS(R4)             ;
    FED0     31       17E0  5484         BRW     PEND                      ; Hold the request pending
                     17E3  5485                                           ;  until DNI is set
                     17E3  5486
                     17E3  5487 IN_HALT:                                  ; Halt the port (DELUA only)
   4E 8F     90      17E3  5488         MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_HALT,- ; Load command
     62               17E6  5489                 PCSR0(R2)                 ;  and enable interrupts
     02     8A        17E7  5490         BICB    #CDB_STS_M_RUN,-          ; Clear the RUN bit
 025A C4               17E9  5491                 CDB_B_STS(R4)             ;
                     17EC  5492         ;BRB    REQ_COM                   ; Request is now complete
                     17EC  5493
                     17EC  5494 ;+
                     17EC  5495 ; Complete request
                     17EC  5496 ;-
                     17EC  5497
                     17EC  5498 REQ_COM:                                  ; Request complete
     50    01    9A   17EC  5499         MOVZBL  S^#SS$_NORMAL,R0          ; Indicate success
                     17EF  5500 REQ_COM_ERR:                              ; Complete request in error
   3A A3    50    B0   17EF  5501         MOVW    R0,IRP$W_XE_STATUS(R3)    ; Set completion status
00B0 D4    63    0E    17F3  5502         INSQUE  (R3),@CDB_Q_POST+4(R4)    ; Insert IRP on posting queue
   53    02 A2    B0   17F8  5503         MOVW    PCSR1(R2),R3              ; Fetch the CSR contents
53   53    10    78    17FC  5504         ASHL    #16,R3,R3                 ; Shift to high word of R3
     53    62    B0    1800  5505         MOVW    PCSR0(R2),R3              ; Fetch remaining CSR contents
        00FE     31    1803  5506         BRW     SCHED_FORK                ; Schedule a fork process
```

```
                                1806  5508          .SBTTL  DONE_INTR - DEUNA/DELUA COMPLETION INTERRUPT SERVICE ROUTINE
                                1806  5509 ;++
                                1806  5510 ; DONE_INTR - DEUNA/DELUA COMPLETION INTERRUPT SERVICE ROUTINE
                                1806  5511 ;
                                1806  5512 ; Functional description:
                                1806  5513 ;
                                1806  5514 ; This routine services the interrupts generated by the DEUNA/DELUA for
                                1806  5515 ; completion of requests.
                                1806  5516 ;
                                1806  5517 ; Inputs:
                                1806  5518 ;
                                1806  5519 ;       00(SP) = ADDRESS OF UNIT IDB ADDRESS
                                1806  5520 ;
                                1806  5521 ;       R0,R1,R2,R3,R4,R5 ARE AT 04(SP) TO 1C(SP)
                                1806  5522 ;
                                1806  5523 ;
                                1806  5524 ;       IPL = DIPL
                                1806  5525 ;
                                1806  5526 ; Outputs:
                                1806  5527 ;
                                1806  5528 ;       THE INTERRUPT IS DISMISSED
                                1806  5529 ;
                                1806  5530 ; IMPLICIT OUTPUTS:
                                1806  5531 ;
                                1806  5532 ;       A fork process is started to check ring entries.
                                1806  5533 ;--
                                1806  5534 DONE_INTR::                                ; DEUNA/DELUA done interrupt
      54    9E    D0            1806  5535          MOVL    @(SP)+,R4                 ; Get IDB address
   55    18 A4    D0            1809  5536          MOVL    IDB$L_UCBLST(R4),R5       ; Get first UCB address
                                180D  5537          ASSUME  IDB$L_CSR EQ 0
      52    64    D0            180D  5538          MOVL    (R4),R2                   ; Get CSR address
54    0180 C5    D0            1810  5539          MOVL    UCB$L_XE_CDB(R5),R4       ; Get CDB address
            34    13            1815  5540          BEQL    5$                        ; Br if CDB not allocated
            00    E1            1817  5541          BBC     #CDB_STS_V_INITED,-       ; Br if NOT inited
      2E 025A C4                1819  5542                  CDB_B_STS(R4),5$          ;
            06    E0            181D  5543          BBS     #CDB_STS_V_ERR,-          ; Br if there was an error
      28 025A C4                181F  5544                  CDB_B_STS(R4),5$          ;
      53    02    A2    B0      1823  5545          MOVW    PCSR1(R2),R3              ; Fetch the CSR contents
   53    53    10    78        1827  5546          ASHL    #16,R3,R3                 ; Shift to high word of R3
      53    62    B0            182B  5547          MOVW    PCSR0(R2),R3              ; Fetch remaining CSR contents
      07 53    09    E1        182E  5548          BBC     #XE_PCSR0_V_FATL,R3,1$    ; Was FATL set (DELUA only) ?
53    FFFF0000 8F    CA        1832  5549          BICL2   #^C^XFFFF,R3              ; If so, then zero PCSR1 bits
51    53    F8 8F    78        1839  5550 1$:      ASHL    #-8,R3,R1                 ; Shift out port command
                                183E  5551 ;
                                183E  5552 ; The interrupt bits can only be cleared by writing one's into them, therefore
                                183E  5553 ; we will write one's into all bits which already have one's.
                                183E  5554 ;
      01 A2    51    90        183E  5555          MOVB    R1,PCSR0+1(R2)            ; Release interrupt interlock
      08 53    07    E0        1842  5556          BBS     #XE_PCSR0_V_INTR,R3,10$   ; Br if valid interrupt
         53    10    C8        1846  5557          BISL2   #XE_PCSR0_M_ILLINT,R3     ; Indicate cause of error
               35    11        1849  5558          BRB     15$                       ; Schedule a fork process
                                184B  5559
            00AA    31        184B  5560 5$:      BRW     INTEXIT                   ; Long branch to INTEXIT
                                184E  5561 ;
                                184E  5562 ; We will check for any errors at all, and then for fatal errors only.
                                184E  5563 ; We do this for speed, since error cases should be the exception.
                                184E  5564 ;
```

```
                53   C700 8F    B3   184E  5565 10$:    BITW    #XE_PCSR0_M_USCI!-      ; Any errors?
                                     1853  5566                 XE_PCSR0_M_FATL!-       ;       ...
                                     1853  5567                 XE_PCSR0_M_PCEI!-       ;       ...
                                     1853  5568                 XE_PCSR0_M_RCBI!-       ;       ...
                                     1853  5569                 XE_PCSR0_M_SERI,R3      ;       ...
                          78   13    1853  5570         BEQL    20$                     ; Br if NOT
                  16 53    0A   E1    1855  5571         BBC     #XE_PCSR0_V_RCBI,R3,13$ ; Br if FATAL error
                           20   A8    1859  5572         BISW    #CDB_STS_M_RCBI,-       ; Indicate RCBI error
                      025A C4         185B  5573                 CDB_B_STS(R4)           ;
                                      185E  5574         INCC    CDB_W_SBUCTR(R4),W      ; BUMP the counter
                53   C300 8F    B3   1868  5575         BITW    #XE_PCSR0_M_USCI!-      ; Is it also a FATAL error?
                                     186D  5576                 XE_PCSR0_M_FATL!-       ;       ...
                                     186D  5577                 XE_PCSR0_M_PCEI!-       ;       ...
                                     186D  5578                 XE_PCSR0_M_SERI,R3      ;       ...
                          5E   13    186D  5579         BEQL    20$                     ; Br if NOT
                53   4300 8F    B3   186F  5580 13$:    BITW    #XE_PCSR0_M_USCI!-      ; Is this a fatal error?
                                     1874  5581                 XE_PCSR0_M_FATL!-       ;       ...
                                     1874  5582                 XE_PCSR0_M_PCEI,R3      ;
                          0A   12    1874  5583         BNEQ    15$                     ; Br if YES
            50    53     F0 8F    78   1876  5584         ASHL    #-16,R3,R0              ; Shift down PCSR1 contents
                      50   03   91    187B  5585         CMPB    #XE_STAT_V_RUN,R0       ; Is this the run state?
                           4D   13    187E  5586         BEQL    20$                     ; Br if yes - ignore error indicator
                                     1880  5587 15$:    SETBIT  #XE_PCSR0_V_USCI,R3     ; Set fatal error indicator
                                     1884  5588         SETBIT  #CDB_STS_V_ERR,-        ; Ignore futher interrupts
                                     1884  5589                 CDB_B_STS(R4)           ;
                0258 C4    08   8A    188A  5590         BICB    #CDB_STS_M_FUNC,CDB_B_STS(R4) ; Indicate function is pending
                     00D6 C4    90    188F  5591         MOVB    CDB_G_PCBB(R4),-        ; Save current PCBB contents
                     00CC C4         1893  5592                 CDB_B_SPARE(R4)         ;  in temporary cell for now
                          0E    B0    1896  5593         MOVW    #PB_FC_V_RDSTATUS,-     ; Request the PORT STATUS
                     00D6 C4         1898  5594                 CDB_G_PCBB(R4)          ;
                       42 8F    90    189B  5595         MOVB    #XE_PCSR0_M_INTE!XE_CMD_V_GETCMD,- ; Fetch the port status
                          62         189E  5596                 PCSR0(R2)               ;
                                     189F  5597         TIMEWAIT #5,#XE_PCSR0_M_DNI,PCSR0(R2),W ; Wait for DNI or 50 usec
                     00CC C4    90    18C4  5598         MOVB    CDB_B_SPARE(R4),-       ; Save previous PCBB contents
                     00D7 C4         18C8  5599                 CDB_G_PCBB+1(R4)        ;  in high byte of PCBB
                          29    11    18CB  5600         BRB     30$                     ; Schedule a fork process
                                     18CD  5601
              25 53    0B    E1    18CD  5602 20$:      BBC     #XE_PCSR0_V_DNI,R3,30$  ; Br if not a completion interrupt
          0258 C4      08    88    18D1  5603         BISB    #CDB_STS_M_FUNC,CDB_B_STS(R4) ; Indicate function complete
                0223 C4    94    18D6  5604         CLRB    CDB_B_TIM_DNI(R4)       ; Stop DNI timer
          52    008C D4    0F    18DA  5605         REMQUE  @CDB_Q_DONE(R4),R2      ; Get waiting packet
                     0C    1D    18DF  5606         BVS     25$                     ; Br if none
                00D6 C4    7D    18E1  5607         MOVQ    CDB_G_PCBB(R4),-        ; Take snapshot of PCBB
                0094 C2         18E5  5608                 IRP$Q_XE_PCB(R2)        ;       ...
          00B0  D4         62    0E    18E8  5609         INSQUE  (R2),@CDB_Q_POST+4(R4)  ; Insert packet on posting queue
                                     18ED  5610 25$:      PUSHQ   R3                      ; Save R3, R4
                     FBC9    30    18F0  5611         BSBW    LOAD_PORT               ; Load CSR's with next command
                                     18F3  5612         POPQ    R3                      ; Restore R3, R4
                     0C      10    18F6  5613 30$:      BSBB    SCHED_FORK              ; Schedule a fork process
                                     18F8  5614
                                     18F8  5615 INTEXIT:                              ; Exit interrupt
                50   8E      7D    18F8  5616         MOVQ    (SP)+,R0                ; Restore Regs
                52   8E      7D    18FB  5617         MOVQ    (SP)+,R2
                54   8E      7D    18FE  5618         MOVQ    (SP)+,R4
                                     1901  5619         REI                             ; Dismiss the interrupt
```

```
                     1902  5621          .SBTTL  SCHED_FORK - SCHEDULE THE FORK PROCESS
                     1902  5622          .SBTTL  SCHED_FORKC - SCHEDULE THE FORK PROCESS WITH R3 CLEAR
                     1902  5623 ;++
                     1902  5624 ; SCHED_FORK - Schedule the fork process
                     1902  5625 ; SCHED_FORKC - Schedule the fork process with R3 clear
                     1902  5626 ;
                     1902  5627 ; Functional description:
                     1902  5628 ;
                     1902  5629 ; This routine is called to schedule the error and I/O completion fork process.
                     1902  5630 ; The last controller port and CSR values are saved for examination.  If the
                     1902  5631 ; fork process is already pending, only the last CSR values are saved if there
                     1902  5632 ; was an error.
                     1902  5633 ;
                     1902  5634 ; Inputs:
                     1902  5635 ;
                     1902  5636 ;       R3 = Last PCSR0 and PCSR1 values
                     1902  5637 ;       R4 = CDB address
                     1902  5638 ;
                     1902  5639 ;       IPL = DIPL or higher
                     1902  5640 ;
                     1902  5641 ; Outputs:
                     1902  5642 ;
                     1902  5643 ;       R3 is cleared if SCHED_FORKC entry.
                     1902  5644 ;       R4 is destroyed.
                     1902  5645 ;       R5 = CDB address
                     1902  5646 ;
                     1902  5647 ; If XE_PCSR0_V_USCI is set in PCSR0, then the following is returned:
                     1902  5648 ;
                     1902  5649 ;       CDB_L_PCSR0(R4) = PCSR0 and PCSR1 values
                     1902  5650 ;
                     1902  5651 ;--
                     1902  5652
                     1902  5653 SCHED_FORKC::                              ; Schedule fork process, clr R3
          53    D4   1902  5654          CLRL    R3                        ; No device error
                     1904  5655 SCHED_FORK::                               ; Schedule fork process
 0C 025A C4    02    E2   1904  5656          BBSS    #CDB_STS_V_FORK_PEND,CDB_B_STS(R4),10$ ; Br if fork pending
                     190A  5657          ASSUME  CDB_C_FQFL EQ 0
       55    54    D0   190A  5658          MOVL    R4,R5                     ; Get CDB fork block address
         21'AF    9F   190D  5659          PUSHAB  B^FORK_PROC               ; Else, set address of fork process
    00000000'GF    17   1910  5660          JMP     G^EXE$FORK                ; Schedule the fork and return
                     1916  5661
    04 53    08    E1   1916  5662 10$:     BBC     #XE_PCSR0_V_USCI,R3,20$ ; Br if not error
    10 A4    53    D0   191A  5663          MOVL    R3,CDB_L_PCSR0(R4)        ; Save last CSR values
                05    191E  5664 20$:     RSB                               ; Return to caller
```

```
                                        .SBTTL  FORK_PROC - Error and completion fork process handling
                                ;++
                                ; FORK_PROC - Error and completion fork processing
                                ;
                                ; Functional description:
                                ;
                                ; This routine is called as a fork process to handle errors and all completions
                                ; pending.
                                ;
                                ; Inputs:
                                ;
                                ;       R3 = Last SEL0 and SEL2 values
                                ;       R4 = CDB address
                                ;       R5 = CDB address
                                ;
                                ;       IPL = FIPL
                                ;
                                ; Outputs:
                                ;
                                ;       R0-R5 are destroyed.
                                ;
                                ;--
                        0A0E'   .WORD   TIMEOUT-.               ; Offset to timeout routine
                                .ENABL  LSB
                                FORK_PROC::                     ; Error/completion fork process
                                CLRBIT  #CDB_STS_V_FORK_PEND,CDB_B_STS(R4) ; Clear fork process flag
          1C 53   08    E1      BBC     #XE_PCSR0_V_USCI,R3,10$ ; Br if not an error
                                SETBIT  #XM$V_ERR_FATAL,CDB_L_DEVDEPEND(R4) ; Indicate fatal error
    55    53    F8 8F    78      ASHL    #-8,R3,R5               ; Get high byte of PCSR0
       00C7 C4    55    90      MOVB    R5,CDB_L_DEVDEPEND+3(R4) ; Save high byte of PCSR0
       55   00C8 C4    D0      MOVL    CDB_L_UCB0(R4),R5       ; Get UCB #0 address
            0082 C5    B6      INCW    UCB$W_ERRCNT(R5)        ; Bump error counter
               0A81    31      BRW     SHUTDOWN_UNA            ; Shutdown the DEUNA/DELUA device
                                ;
                                ; Complete any requests on completion queue
                                ;
    53    00AC D4    0F      10$:    REMQUE  @CDB_Q_POST(R4),R3      ; Get next completed block
             04    1C              BVC     20$                     ; Br if got one
            01F3   30              BSBW    SCAN_LISTS              ; Scan XMIT/RECV lists
                   05              RSB                             ; Return to caller
                                ;
                                ; Found a completed block - finish it
                                ;
      0A    0A A3    91      20$:    CMPB    IRP$B_TYPE(R3),S^#DYN$C_IRP ; Was it an IRP?
             04    13              BEQL    30$                     ; Br if yes - complete it
                                   BUG_CHECK NOBUFPCKT,FATAL       ; Else, fatal error
                                ;
                                ; IRP packet completion - branch to proper completion handling routine.
                                ;
    52    21 A3    9A      30$:    MOVZBL  IRP$B_XE_FUNC(R3),R2    ; Get function type
    55    1C A3    D0              MOVL    IRP$L_UCB(R3),R5        ; Set UCB address
                                   $DISPATCH       R2,TYPE=B,-
                                   <-      ;function       action

                                           <XE_FC_V_INIT    FORK_INIT>,-    ; Start request
                                           <XE_FC_V_RESTART FORK_INIT>,-    ; Automatic restart
                                           <XE_FC_V_TESTD   FORK_INIT>,-    ; Self-test done
```

```
                     1964  5723                <XE_FC_V_LDRING FORK_INIT>,-     ; Load ring pointer
                     1964  5724                <XE_FC_V_RDDPA  FORK_INIT>,-     ; Read default physical address
                     1964  5725                <XE_FC_V_WTPHYAD FORK_INIT>,-    ; Write physical address
                     1964  5726                <XE_FC_V_WTMULTI FORK_INIT>,-    ; Write multicast address
                     1964  5727                <XE_FC_V_WTMODE FORK_INIT>,-     ; Write DEUNA/DELUA mode
                     1964  5728                <XE_FC_V_START  FORK_INIT>,-     ; Start XMIT/RECV process
                     1964  5729                <XE_FC_V_RDCNTS FORK_COUNTER>,-  ; Read counts
                     1964  5730                <XE_FC_V_RCCNTS FORK_COUNTER>,-  ; Read and clear counts
                     1964  5731                <XE_FC_V_MULTI  FORK_PKT>,-      ; Write multicast address list
                     1964  5732                <XE_FC_V_LDMEM  FORK_LDMEM>,-    ; Load DEUNA/DELUA memory
                     1964  5733                <XE_FC_V_RDMEM  FORK_RDMEM>,-    ; Read DEUNA/DELUA memory
                     1964  5734                <XE_FC_V_UPST   FORK_PKT>,-      ; Load and start micro-address
                     1964  5735                <XE_FC_V_LDSID  FORK_PKT>,-      ; Load system ID parameters
                     1964  5736                <XE_FC_V_RDSID  FORK_RDSID>,-    ; Read system ID parameters
                     1964  5737                <XE_FC_V_STOPPR FORK_PKT>,-      ; Stop xmit and receive process
                     1964  5738                <XE_FC_V_STEST  FORK_PKT>,-      ; Perform DEUNA/DELUA self test
                     1964  5739                <XE_FC_V_READST FORK_READST>,-   ; Read port status
                     1964  5740                <XE_FC_V_CHMODE FORK_PKT>,-      ; Change the mode definition
                     1964  5741                <XE_FC_V_STOP   FORK_STOP>,-     ; Stop protocol
                     1964  5742                <XE_FC_V_MOPCTR FORK_PKT>,-      ; Mop counter read request
                     1964  5743                <XE_FC_V_ESTEST FORK_PKT>,-      ; Perform DELUA extended self test
                     1964  5744                <XE_FC_V_HALT   FORK_PKT>,-      ; Halt port (DELUA only)
                     1964  5745            >
                     19A0  5746 ;
                     19A0  5747 ; Otherwise BUG_CHECK if not a valid IRP
                     19A0  5748 ;
                     19A0  5749            BUG_CHECK NOBUFPCKT,FATAL           ; Fatal driver error
                     19A4  5750
                     19A4  5751 FORK_PKT:                                      ; Process completion on packet
     50   3A A3   3C 19A4  5752            MOVZWL  IRP$W_XE_STATUS(R3),R0      ; Get status of completion
           05A0   30 19A8  5753            BSBW    IO_DONE                     ; Complete the I/O
           FF99   31 19AB  5754            BRW     10$                         ; Get next packet
                     19AE  5755
                     19AE  5756 FORK_INIT:                                     ; Process init completion
     39 A3   00   91 19AE  5757            CMPB    S^#XE_FC_V_INIT,IRP$L_MEDIA+1(R3) ; Was this an initialize?
              F0   12 19B2  5758            BNEQ    FORK_PKT                    ; Br if not
     50   3A A3   3C 19B4  5759            MOVZWL  IRP$W_XE_STATUS(R3),R0      ; Get completion status
              50   DD 19B8  5760            PUSHL   R0                          ; Save completion status
        50    14   B1 19BA  5761            CMPW    #SS$_BADPARAM,R0            ; Is this bad parameter?
              0E   12 19BD  5762            BNEQ    35$                         ; Br if no
                     19BF  5763 ;
                     19BF  5764 ; For SS$_BADPARAM, must return bad parameter code
                     19BF  5765 ;
     38 A3   50   3C 19BF  5766            MOVZWL  R0,IRP$L_IOST1(R3)          ; Return error code
3C   A3   40 A3   3C 19C3  5767            MOVZWL  IRP$W_XE_CODE(R3),IRP$L_IOST2(R3) ; Return bad parameter code
           059A   30 19C8  5768            BSBW    IO_DONE2                    ; Complete the I/O request
             03   11 19CB  5769            BRB     37$                         ; On our merry way...
                     19CD  5770 ;
                     19CD  5771 ; Not SS$_BADPARAM error code, no bad parameter code
                     19CD  5772 ;
           057B   30 19CD  5773 35$:       BSBW    IO_DONE                     ; Complete the I/O request
                     19D0  5774
              50 8ED0 19D0  5775 37$:       POPL    R0                          ; Restore status
        03    50   E9 19D3  5776            BLBC    R0,40$                      ; Br ir error on startup
           F9D4   30 19D6  5777            BSBW    FILLRCVLIST                 ; Start the receives on success
           FF6B   31 19D9  5778 40$:       BRW     10$                         ; Get next packet
                     19DC  5779
```

```
                                19DC  5780 FORK_STOP:
                     0B5B   30  19DC  5781          BSBW    SHUTDOWN_PROTYP          ; Shutdown the PROTOCOL
                  50   01   9A  19DF  5782          MOVZBL  S^#SS$_NORMAL,R0         ; Return success
                     0566   31  19E2  5783          BRW     IO_DONE                  ; Complete the I/O request
                                19E5  5784
                                19E5  5785 FORK_COUNTER:                            ; Process counter done
                  00F0 8F   BB  19E5  5786          PUSHR   #^M<R4,R5,R6,R7>         ; Save registers
                                19E9  5787          ASSUME  P2B_L_POINTER EQ 0
                52   2C B3   D0  19E9  5788          MOVL    @IRP$L_SVAPTE(R3),R2     ; Point to start of data area
             56   00EA C4   9E  19ED  5789          MOVAB   CDB_G_UDB(R4),R6         ; Get address of counters
             55   E733 CF   9E  19F2  5790          MOVAB   CTRTAB,R5                ; Get address of counter format table
                  51   11   9A  19F7  5791          MOVZBL  #LINE_CNT_SIZE,R1        ; Get number of entries in table
                  50   85   3C  19FA  5792 50$:     MOVZWL  (R5)+,R0                 ; Get counter code
                  82   50   B0  19FD  5793          MOVW    R0,(R2)+                 ; Return counter type code
                  57   85   3C  1A00  5794          MOVZWL  (R5)+,R7                 ; Get offset word
          50   50   03   0C   EF  1A03  5795          EXTZV   #NMA$V_CNT_MAP,#3,R0,R0  ; Get width + bit map indicator
                                1A08  5796          CASE    R0,TYPE=B,LIMIT=#2,<-    ; Dispatch on width and bit map
                                1A08  5797                  60$,-                    ; 8 bit counter
                                1A08  5798                  70$,-                    ; 8 bit counter + bit map
                                1A08  5799                  80$,-                    ; 16 bit counter
                                1A08  5800                  90$,-                    ; 16 bit counter + bit map
                                1A08  5801                  100$>                    ; 32 bit counter
                                1A16  5802 ;
                                1A16  5803 ; All others - fatal error
                                1A16  5804 ;
                                1A16  5805          BUG_CHECK NOBUFPCKT,FATAL
                                1A1A  5806 ;
                                1A1A  5807 ; 8 bit counter
                                1A1A  5808 ;
                82   6647   90  1A1A  5809 60$:     MOVB    (R6)[R7],(R2)+           ; Store counter value
                     20   11  1A1E  5810          BRB     110$                     ; Check if more
                                1A20  5811 ;
                                1A20  5812 ; 8 bit counter + bit map
                                1A20  5813 ;
             82   6647   90  1A20  5814 70$:     MOVB    (R6)[R7],(R2)+           ; Store bit map
                  57   D6  1A24  5815          INCL    R7                       ; Skip to next byte
             82   6647   90  1A26  5816          MOVB    (R6)[R7],(R2)+           ; Store rest of bit map
                  57   D6  1A2A  5817          INCL    R7                       ; Skip to next byte
                  EC   11  1A2C  5818          BRB     60$                      ; Store counter value
                                1A2E  5819 ;
                                1A2E  5820 ; 16 bit counter
                                1A2E  5821 ;
             82   6647   B0  1A2E  5822 80$:     MOVW    (R6)[R7],(R2)+           ; Store counter value
                  0C   11  1A32  5823          BRB     110$                     ; Check if more
                                1A34  5824 ;
                                1A34  5825 ; 16 bit counter + bit map
                                1A34  5826 ;
             82   6647   B0  1A34  5827 90$:     MOVW    (R6)[R7],(R2)+           ; Store bit map
                  57   D6  1A38  5828          INCL    R7                       ; Skip to counter
                  F2   11  1A3A  5829          BRB     80$                      ; Store counter value
                                1A3C  5830 ;
                                1A3C  5831 ; 32 bit counter
                                1A3C  5832 ;
             82   6647   D0  1A3C  5833 100$:    MOVL    (R6)[R7],(R2)+           ; Store counter value
                                1A40  5834          ;BRB    110$                     ; Check if more
                                1A40  5835
                B7 51   F5  1A40  5836 110$:    SOBGTR  R1,50$                   ; Loop if more
```

```
                                1A43  5837 ;
                                1A43  5838 ; Get the counters kept by the driver
                                1A43  5839 ;
          55    E726 CF    9E   1A43  5840          MOVAB   LINE_CTR,R5             ; Get address of counter format table
                51   03    9A   1A48  5841          MOVZBL  #LINE_CTR_SIZE,R1       ; Get number of entries in table
                50   85    3C   1A4B  5842 112$:    MOVZWL  (R5)+,R0                ; Get counter code
                82   50    B0   1A4E  5843          MOVW    R0,(R2)+                ; Return counter type code
                57   85    3C   1A51  5844          MOVZWL  (R5)+,R7                ; Get offset word
                57   54    C0   1A54  5845          ADDL    R4,R7                   ; Point to counter in CDB
       50   50   03   0C    EF   1A57  5846          EXTZV   #NMA$V_CNT_MAP,#3,R0,R0 ; Get width + bit map indicator
                                1A5C  5847          CASE    R0,TYPE=B,LIMIT=#2,<-   ; Dispatch on width and bit map
                                1A5C  5848                  113$,-                  ; 8 bit counter
                                1A5C  5849                  113$,-                  ; 8 bit counter + bit map
                                1A5C  5850                  114$,-                  ; 16 bit counter
                                1A5C  5851                  113$,-                  ; 16 bit counter + bit map
                                1A5C  5852                  113$>                   ; 32 bit counter
                                1A6A  5853
                                1A6A  5854 113$:    BUG_CHECK NOBUFPCKT,FATAL       ; All others, fatal error
                                1A6E  5855 ;
                                1A6E  5856 ; 16 BIT counter
                                1A6E  5857 ;
                82   67    B0   1A6E  5858 114$:    MOVW    (R7),(R2)+              ; Store counter in buffer
                     0B    91   1A71  5859          CMPB    #XE_FC_V_RCCNTS,-       ; Is this a clear operation
                  21 A3         1A73  5860                  IRP$B_XE_FUNC(R3)       ;  as well?
                     02    12   1A75  5861          BNEQ    115$                    ; Br if not
                     67    B4   1A77  5862          CLRW    (R7)                    ; Else, clear the counter
                CF   51    F5   1A79  5863 115$:    SOBGTR  R1,112$                 ; Loop if more
                                1A7C  5864
              00F0   8F    BA   1A7C  5865          POPR    #^M<R4,R5,R6,R7>        ; Restore registers
           50   32 A3    3C   1A80  5866          MOVZWL  IRP$W_BCNT(R3),R0       ; Get size of P2 buffer
                32 A3    B4   1A84  5867          CLRW    IRP$W_BCNT(R3)          ; Assume error - no data return
             18 3A A3    E9   1A87  5868          BLBC    IRP$W_XE_STATUS(R3),130$        ; Br if err when getting counts
           50   38 A3    B1   1A8B  5869          CMPW    IRP$W_XE_USERSIZ(R3),R0 ; Is user buffer larger than P2?
                   0A    1E   1A8F  5870          BGEQU   120$                    ; Br if yes - okay
           50   38 A3    B0   1A91  5871          MOVW    IRP$W_XE_USERSIZ(R3),R0 ; Else, set user return size
    3A A3    0601   8F    B0   1A95  5872          MOVW    #SS$_BUFFEROVF,IRP$W_XE_STATUS(R3) ; Set error return
       32 A3    50   B0   1A9B  5873 120$:    MOVW    R0,IRP$W_BCNT(R3)       ; Set user return size
     50   50    10   78   1A9F  5874          ASHL    #16,R0,R0               ; Move count to high word
       50   3A A3    B0   1AA3  5875 130$:    MOVW    IRP$W_XE_STATUS(R3),R0  ; Set return status
               04A1   30   1AA7  5876          BSBW    IO_DONE                 ; Complete the I/O request
                FE9A  31   1AAA  5877          BRW     10$                     ; Get next packet
                                1AAD  5878
                                1AAD  5879 FORK_RDMEM:                          ; Process READ DEUNA/DELUA MEMORY
           50   32 A3    3C   1AAD  5880          MOVZWL  IRP$W_BCNT(R3),R0       ; Get size of P2 buffer
                32 A3    B4   1AB1  5881          CLRW    IRP$W_BCNT(R3)          ; Assume error - no data return
             04 3A A3    E9   1AB4  5882          BLBC    IRP$W_XE_STATUS(R3),140$        ; Br if err when getting counts
       32 A3    50   B0   1AB8  5883          MOVW    R0,IRP$W_BCNT(R3)       ; Else, reset buffer size
     50   50    10   78   1ABC  5884 140$:    ASHL    #16,R0,R0               ; Get return buffer size
       50   3A A3    B0   1AC0  5885          MOVW    IRP$W_XE_STATUS(R3),R0  ; Set return status
                04   11   1AC4  5886          BRB     145$                    ; Complete the request
                                1AC6  5887
                                1AC6  5888 FORK_LDMEM:                          ; Process LOAD DEUNA/DELUA MEMORY
           50   3A A3    3C   1AC6  5889          MOVZWL  IRP$W_XE_STATUS(R3),R0  ; Get completion status
           52   24 A5    D0   1ACA  5890 145$:    MOVL    UCB$L_CRB(R5),R2        ; Get CRB address
                                1ACE  5891          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                                1ACE  5892          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
                3C A3    D0   1ACE  5893          MOVL    IRP$L_XE_MAP(R3),-      ; Setup map register data
```

```
                34 A2          1AD1  5894                    CRB$L_INTD+VEC$W_MAPREG(R2) ;   ...
                 0475    30    1AD3  5895            BSBW    IO_DONE                     ; Complete the request
                         1AD6  5896            RELMPR                                    ; Release map registers
                 FE68    31    1ADC  5897            BRW     10$                         ; Look for more work
                               1ADF  5898
                               1ADF  5899 FOR:_RDSID:                                    ; Process READ SYSTEM ID PARAMETERS
          50    32 A3    3C    1ADF  5900            MOVZWL  IRP$W_BCNT(R3),R0           ; Get size of P2 buffer
                32 A3    B4    1AE3  5901            CLRW    IRP$W_BCNT(R3)              ; Assume error - no data return
                  28 3A A3    E9  1AE6  5902            BLBC    IRP$W_XE_STATUS(R3),160$        ; Br if err when getting parameters
                      3C    B0    1AEA  5903            MOVW    #UDB_SID_C_LENGTH,-         ; Assume user buffer is large enough
                32 A3           1AEC  5904                    IRP$W_BCNT(R3)              ;
                   3C     50    B1    1AEE  5905            CMPW    R0,#UDB_SID_C_LENGTH        ; Is user buffer larger than UDB ?
                          0A    1E    1AF1  5906            BGEQU   150$                        ; Br if yes - okay
             32 A3        50    B0    1AF3  5907            MOVW    R0,IRP$W_BCNT(R3)           ; Else, set buffer size to user buffer
3A A3             0601    8F    B0    1AF7  5908            MOVW    #SS$_BUFFEROVF,IRP$W_XE_STATUS(R3) ; Set error return
                          38    BB    1AFD  5909 150$:      PUSHR   #^M<R3,R4,R5>               ; Save registers
                               1AFF  5910            ASSUME  P2B_L_POINTER EQ 0
          52    2C B3          D0    1AFF  5911            MOVL    @IRP$L_SVAPTE(R3),R2        ; Point to start of data area
       54    016E C4           9E    1B03  5912            MOVAB   CDB_G_SIDUDB(R4),R4         ; Get address of SYSTEM ID PARAMETERS
          50    32 A3          3C    1B08  5913            MOVZWL  IRP$W_BCNT(R3),R0           ; Get size of transfer
       62    64    50          28    1B0C  5914            MOVC3   R0,(R4),(R2)                ; Move data to system buffer
                          38    BA    1B10  5915            POPR    #^M<R3,R4,R5>               ; Restore registers
50    32 A3           10    78    1B12  5916 160$:      ASHL    #16,IRP$W_BCNT(R3),R0       ; Get return buffer size
          50    3A A3          B0    1B17  5917            MOVW    IRP$W_XE_STATUS(R3),R0      ; Set return status
                   042D    30    1B1B  5918            BSBW    IO_DONE                     ; Complete the request
                   FE26    31    1B1E  5919            BRW     10$                         ; Look for more
                               1B21  5920
                               1B21  5921 FORK_READST:                                   ; Process READ PORT STATUS
                32 A3          B4    1B21  5922            CLRW    IRP$W_BCNT(R3)              ; Assume error - no data return
             0D 3A A3          E9    1B24  5923            BLBC    IRP$W_XE_STATUS(R3),180$        ; Br if err when getting status
          32 A3            08    B0    1B28  5924            MOVW    #8,IRP$W_BCNT(R3)           ; Assume user buffer is large enough
                               1B2C  5925            ASSUME  P2B_L_POINTER EQ 0
          52    2C B3          D0    1B2C  5926            MOVL    @IRP$L_SVAPTE(R3),R2        ; Point to start of data area
       62    0094 C3           7D    1B30  5927            MOVQ    IRP$O_XE_PCB(R3),(R2)       ; Copy return data
50    32 A3           10    78    1B35  5928 180$:      ASHL    #16,IRP$W_BCNT(R3),R0       ; Get return buffer size
          50    3A A3          B0    1B3A  5929            MOVW    IRP$W_XE_STATUS(R3),R0      ; Set return status
                   040A    30    1B3E  5930            BSBW    IO_DONE                     ; Complete the request
                   FE03    31    1B41  5931            BRW     10$                         ; Look for more
                               1B44  5932            .DSABL  LSB
```

```
                     1B44  5934          .SBTTL  SCAN_LISTS - Scan transmit and receive ring entries
                     1B44  5935 ;++
                     1B44  5936 ; SCAN_LISTS - Scan transmit and recieve ring entries for completed requests
                     1B44  5937 ;
                     1B44  5938 ; Functional description:
                     1B44  5939 ;
                     1B44  5940 ; This routine is called to search the ring entries for completed operations.
                     1B44  5941 ;
                     1B44  5942 ; Inputs:
                     1B44  5943 ;
                     1B44  5944 ;       R4 = CDB address
                     1B44  5945 ;
                     1B44  5946 ;       IPL = FIPL
                     1B44  5947 ;
                     1B44  5948 ; Outputs:
                     1B44  5949 ;
                     1B44  5950 ;       R0-R3,R5 are destroyed.
                     1B44  5951 ;       All other registers are preserved.
                     1B44  5952 ;
                     1B44  5953 ;--
                     1B44  5954
                     1B44  5955 SCAN_LISTS::                                 ; Scan XMIT/RECV rings
          56   DD    1B44  5956          PUSHL   R6                         ; Save R6
     00B6 C4   95    1B46  5957 10$:     TSTB    CDB_B_RCVCNT(R4)           ; Any receives in progress?
          03   12    1B4A  5958          BNEQ    20$                        ; Br if yes - process any receives
        0118   31    1B4C  5959 15$:     BRW     60$                        ; Else, skip to the xmit list
                     1B4F  5960
        02B4 D4   16 1B4F  5961 20$:     JSB     @CDB_L_ASSPKT(R4)          ; Assemble receive packets
          F6 50   E9 1B53  5962          BLBC    R0,15$                     ; Br on error or none
          06   E0   1B56  5963          BBS     #RCV_FLG_V_ERRS,-          ; Br if FATAL receive error
                     1B58  5964                   CXB$B_R_FLAGS(R2),25$      ;   DEUNA/DELUA should count this error
        3C 0B A2
     51   44 A2   3C 1B5B  5965          MOVZWL  CXB$W_R_PTYPE(R2),R1       ; Pick up protocol type from buffer
                     1B5F  5966
                     1B5F  5967          .IF DF  POINT
0660 8F    51   B1 1B5F  5968          CMPW    R1,#XE_C_STPRO             ;% Is this the startup protocol?
          08   12   1B64  5969          BNEQ    21$                        ;% Br if not
        0EAA   30   1B66  5970          BSBW    FIND_POINT_UCB             ;% Find the point to point user!
       14 50   E9 1B69  5971          BLBC    R0,22$                     ;% Br if failure, assume regular user
          29   11   1B6C  5972          BRB     25$                        ;% Else done with MSG block
                     1B6E  5973          .ENDC
                     1B6E  5974
0260 8F    51   B1 1B6E  5975 21$:     CMPW    R1,#NI_CTR_PROTYP          ; Is this the Remote Console protocol?
          08   12   1B73  5976          BNEQ    22$                        ; Br if not
          09   91   1B75  5977          CMPB    #NI_CTR_READ,-             ; Is this a read counters request?
       46 A2        1B77  5978                   CXB$T_R_USERDAT(R2)        ;
          05   12   1B79  5979          BNEQ    22$                        ; Br if not
        0592   30   1B7B  5980          BSBW    MOP_CTR_REQUEST            ; Else, process the request for counters
          C6   11   1B7E  5981          BRB     10$
        157C   30   1B80  5982 22$:     BSBW    MATCH_PROTYP               ; Try to match protocol type
       1E 50   E8 1B83  5983          BLBS    R0,27$                     ; Br if success
55   0224 C4   D0 1B86  5984 23$:     MOVL    CDB_L_PRMUSER(R4),R5       ; Try to get the PROMISCUOUS user
          5F   12   1B8B  5985          BNEQ    34$                        ; Br if one found
                     1B8D  5986          INCC    CDB_W_UFDCTR(R4),W         ; Else, no such protocol type
       10 A2   DD   1B97  5987 25$:     PUSHL   CXB$L_LINK(R2)             ; Save next in chain
        F812   30   1B9A  5988          BSBW    ADDRCVLIST                 ; Add buffer to receive list
          52 8ED0   1B9D  5989          POPL    R2                         ; Restore next buffer
          F5   12   1BA0  5990          BNEQ    25$                        ; Loop if more
```

```
                A2   11  1BA2  5991          BRB     10$                     ; Look for next completed buffer
                         1BA4  5992 ;
                         1BA4  5993 ; If there is a promiscuous user, then copy the packet for the promiscuous
                         1BA4  5994 ; user. There is a chance that the data received is not for the protocol
                         1BA4  5995 ; type user just found, we will have to re-verify that the destination
                         1BA4  5996 ; address is for our physical address. This is because if we are running
                         1BA4  5997 ; promiscuous mode, then we will receive all packets, including those not
                         1BA4  5998 ; intended for the protocol user.
                         1BA4  5999 ;
                         1BA4  6000 27$:    ASSUME  NMA$C_STATE_ON EQ 0
                         1BA4  6001         ASSUME  NMA$C_STATE_OFF EQ 1
          23 025D C4   E8  1BA4  6002         BLBS    CDB_B_PRM(R4),32$       ; Br if hardware is NOT in promiscuous mode
            10 38 A2   E8  1BA9  6003         BLBS    CXB$G_R_DEST(R2),30$    ; Br if multicast address, this
                         1BAD  6004                                             ;  will be checked later.
                38 A2    D1  1BAD  6005         CMPL    CXB$G_R_DEST(R2),-      ; Is this packet for this protocol
              026F C4        1BB0  6006                 CDB_G_PHYADR(R4)        ;  user?
                 D1    12  1BB3  6007         BNEQ    23$                     ; Br if not, don't copy packet
                3C A2    B1  1BB5  6008         CMPW    CXB$G_R_DEST+4(R2),-    ; Are we sure?
              0273 C4        1BB8  6009                 CDB_G_PHYADR+4(R4)      ;
                 C9    12  1BBB  6010         BNEQ    23$                     ; Br if no, dont't copy packet
                         1BBD  6011 ;
                         1BBD  6012 ;       Copy the packet for the promiscuous user
                         1BBD  6013 ;
                 55    DD  1BBD  6014 30$:    PUSHL   R5                      ; Save user's UCB address
          55     0224 C4    D0  1BBF  6015         MOVL    CDB_L_PRMUSER(R4),R5    ; Get PROMISCUOUS user's UCB address
                 03    13  1BC4  6016         BEQL    31$                     ; Br if none
                   01E2    30  1BC6  6017         BSBW    COPY_RCV                ; Give buffer to promiscous user
                 55 8ED0   1BC9  6018 31$:    POPL    R5                      ; Restore user's UCB address
                         1BCC  6019 ;
                         1BCC  6020 ; If multicast address is the destination, then make sure that multicast
                         1BCC  6021 ; address is in multicast address list for this unit.
                         1BCC  6022 ;
            1C 38 A2     E9  1BCC  6023 32$:    BLBC    CXB$G_R_DEST(R2),34$    ; Br if physical address
                 14A0    30  1BD0  6024         BSBW    MATCH_MULTI             ; Try to match multicast address
                16 50    E8  1BD3  6025         BLBS    R0,34$                  ; Br if success
                         1BD6  6026 33$:    INCC    UCB$W_XE_MNECTR(R5),W   ; Else, multicast not enabled
                         1BE0  6027         INCC    CDB_W_UFDCTR(R4),W      ; Also added in unrecognized frame dest
                   AB    11  1BEA  6028         BRB     25$                     ; Release buffer
                         1BEC  6029 ;
                         1BEC  6030 ; If the user did not request data chaining, then check to make sure he gets
                         1BEC  6031 ; no chained buffers... may be user of Internal IRPs!
                         1BEC  6032 ;
                10 A2    D5  1BEC  6033 34$:    TSTL    CXB$L_LINK(R2)          ; Is this a chained message?
                   05    13  1BEF  6034         BEQL    36$                     ; Br if not
                         1BF1  6035         ASSUME  NMA$C_STATE_ON EQ 0
                         1BF1  6036         ASSUME  NMA$C_STATE_OFF EQ 1
          4A 00DA C5     E8  1BF1  6037         BLBS    UCB$B_XE_DCR(R5),45$    ; Br if chaining not allowed
                         1BF6  6038 ;
                         1BF6  6039 ; If there is a pending receive I/O request, complete it.
                         1BF6  6040 ; Otherwise, queue the buffer and, if enabled, deliver attention AST.
                         1BF6  6041 ;
          50     42 A5    3C  1BF6  6042 36$:    MOVZWL  UCB$W_DEVBUFSIZ(R5),R0  ; Get size of user's max buffer
                         1BFA  6043         ;&&
                         1BFA  6044         ;&& the following code could cause problems for an altstart user,
                         1BFA  6045         ;&& if the altstart user happens to receive a buffer which is 1 or 2
                         1BFA  6046         ;&& bytes longer than they are capable of handling. Only if the
                         1BFA  6047         ;&& protocol is not "padded", because the size check allows for
```

```
                        1BFA  6048          ;&& 2 bytes of count to be subtracted from the message size.
                        1BFA  6049          ;&&
                        1BFA  6050 ;
                        1BFA  6051 ; Check the size of the received buffer against what the user protocol can
                        1BFA  6052 ; handle.
                        1BFA  6053 ;
                   A3   1BFA  6054          SUBW3   #XE_C_HEADER+-              ; Get the size of the
                        1BFB  6055                  XE_C_CRC+XE_C_CNTSIZ,-      ;  receive buffer
 51    1A A2    14      1BFB  6056                  CXB$Q_BCNT(R2),R1           ;  minus the header and trailer
      50   51    B1     1BFF  6057          CMPW    R1,R0                       ; Is the received size larger than
                        1C02  6058                                              ;  what the user can handle?
           3C    1A     1C02  6059          BGTRU   45$                         ; Br if yes, error
       0178 C5   D5     1C04  6060          TSTL    UCB$L_XE_FFI(R5)            ; FAST Interface supported?
            06   13     1C08  6061          BEQL    37$                         ; Br if not, standard interface
          0226  30      1C0A  6062          BSBW    FINISH_RCV_FFI              ; Else, complete FAST receive
          FF36  31      1C0D  6063          BRW     10$                         ; Look for more completions
                        1C10  6064
    51  00A8 C5   9E    1C10  6065 37$:     MOVAB   UCB$Q_XE_RCVREQ(R5),R1      ; Assume that we are running in
                        1C15  6066                                              ;  exclusive mode
            03   E1     1C15  6067          BBC     #UCB$V_XE_SHARE,-           ; Br if UCB is NOT in SHARED mode
        09 68 A5        1C17  6068                   UCB$W_DEVSTS(R5),38$       ;
                        1C1A  6069 ;
                        1C1A  6070 ; If running in SHARED mode, then we must use the listheads in the SHR
                        1C1A  6071 ; data structure. We will use the source address from the received message
                        1C1A  6072 ; to match against the SHR structure destination address.
                        1C1A  6073 ;
            015A  30    1C1A  6074          BSBW    MATCH_SRC                   ; Check for a match on source address
              B7  12    1C1D  6075          BNEQ    33$                         ; Br if no shared user found
                        1C1F  6076 ;
                        1C1F  6077 ;        SHARED user found, use listheads in SHR data structure.
                        1C1F  6078 ;
    51    20 A1   9E    1C1F  6079          MOVAB   SHR_Q_RCVREQ(R1),R1         ; Get address of waiting IRPs
    53    00 B1   0F    1C23  6080 38$:     REMQUE  @(R1),R3                    ; Remove waiting IRP
               0B   1D    1C27  6081          BVS     40$                         ; Br if none - queue for later
 00C8 C5      50   A2    1C29  6082          SUBW    R0,UCB$W_XE_QUOTA(R5)       ; Else, lessen quota so it can be
                        1C2E  6083                                              ;..increased on completion
            023D  30    1C2E  6084          BSBW    FINISH_RCV_IO               ; And finish the I/O
            FF12  31    1C31  6085          BRW     10$                         ; Look for next completion
                        1C34  6086 ;
                        1C34  6087 ; Check buffer quota and queue if quota okay.
                        1C34  6088 ;
 00C8 C5      50   A2    1C34  6089 40$:     SUBW    R0,UCB$W_XE_QUOTA(R5)       ; Decrement the quota
               1C   1E    1C39  6090          BGEQU   50$                         ; Br if we can buffer request
 00C8 C5      50   A0    1C3B  6091          ADDW    R0,UCB$W_XE_QUOTA(R5)       ; Replace quota
                        1C40  6092 45$:     INCC    UCB$W_XE_UBDCTR(R5),W       ; Else, no buffer available
                        1C4A  6093          INCC    CDB_W_UBDCTR(R4),W          ; ...don't forget CDB counter
            FF40  31    1C54  6094          BRW     25$                         ; Return buffer
                        1C57  6095
                        1C57  6096 50$:     ASSUME  UCB$Q_XE_RCVREQ EQ UCB$Q_XE_RCVMSG+8
                        1C57  6097          ASSUME  SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
        51    04   C2    1C57  6098          SUBL    #4,R1                       ; Backup to backward link pointer
                        1C5A  6099                                              ;  of the message queue
 00 B1    62   0E    1C5A  6100          INSQUE  (R2),@(R1)                  ; Queue received msg for later
            F74C  30    1C5E  6101          BSBW    FILLRCVLIST                 ; Try to fill the receive list
            1466  30    1C61  6102          BSBW    POKE_USER                   ; Deliver ASTs
            FEDF  31    1C64  6103          BRW     10$                         ; Look for more completions
                        1C67  6104 ;
```

```
                     1C67  6105 ; NOW - scan the xmit ring entries
                     1C67  6106 ;
          00B7 C4    95  1C67  6107 60$:    TSTB    CDB_B_XMTCNT(R4)          ; Any xmits in progress?
               03    12  1C6B  6108         BNEQ    75$                       ; Br if yes - look for any completed
             0103    31  1C6D  6109 70$:    BRW     190$                      ; Else, all done
                     1C70  6110
    56    00B4 C4    9A  1C70  6111 75$:    MOVZBL  CDB_B_LASTXMT(R4),R6      ; Get last ring entry completed
    56    4C A446    D0  1C75  6112         MOVL    CDB_L_XRADDR(R4)[R6],R6   ; Retrieve last ring entry address
               07    E0  1C7A  6113         BBS     #XMT_FLG_V_OWN,-          ; Br if not done
          EE 05 A6        1C7C  6114          XMT_B_FLAGS(R6),70$              ;
    53    0094 D4    0F  1C7F  6115         REMQUE  @CDB_Q_XMTPND(R4),R3      ; Get next XMIT CXB
               E7    1D  1C84  6116         BVS     70$                       ; Br if none there (yet)
               04    12  1C86  6117         BNEQ    80$                       ; Br if more entries on queue
          0222 C4    94  1C88  6118         CLRB    CDB_B_TIM_XMT(R4)         ; Stop the xmit timer
          00B7 C4    97  1C8C  6119 80$:    DECB    CDB_B_XMTCNT(R4)          ; One less transmit pending
          00B4 C4    96  1C90  6120         INCB    CDB_B_LASTXMT(R4)         ; Bump ring pointer
            FC 8F    8A  1C94  6121         BICB    #^C<MAX_C_XMT-1>,-        ; Modulo receive ring entry size
          00B4 C4        1C97  6122                 CDB_B_LASTXMT(R4)         ;       ...
                         1C9A  6123
 22 A3      08 A6    B1  1C9A  6124         CMPW    XMT_W_RID(R6),CXB$W_XE_RID(R3) ; Is this a match???
               04    13  1C9F  6125         BEQL    90$                       ; Br if yes, good
                         1CA1  6126         BUG_CHECK NOBUFPCKT,FATAL         ; Else, fatal error
                         1CA5  6127 ;
                         1CA5  6128 ; Transmit complete
                         1CA5  6129 ;
            05 A6    90  1CA5  6130 90$:    MOVB    XMT_B_FLAGS(R6),-         ; Save diagnostic return info
          00CD C4        1CA8  6131          CDB_B_DIAG1(R4)                  ;
            06 A6    B0  1CAB  6132         MOVW    XMT_Q_TDRERR(R6),-        ; Save diagnostic return info
          00CE C4        1CAE  6133          CDB_Q_DIAG2(R4)                  ;
       55   18 A3    3C  1CB1  6134         MOVZWL  CXB$Q_BOFF(R3),R5         ; Get offset to start of data
       55      53    C0  1CB5  6135         ADDL    R3,R5                     ; Point to header
       55   08 A5    D0  1CB8  6136         MOVL    XBUF_L_UCB(R5),R5         ; Get UCB address
       52   22 A3    9A  1CBC  6137         MOVZBL  CXB$B_XE_SLOT(R3),R2      ; Get mapping slot number used
                         1CC0  6138         CLRBIT  R2,CDB_B_XMTMAP(R4)       ; Clear in use flag
       51   24 A5    D0  1CC5  6139         MOVL    UCB$L_CRB(R5),R1          ; Get CRB address
                         1CC9  6140         ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                         1CC9  6141         ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
 34 A1    3C A442    D0  1CC9  6142         MOVL    CDB_L_XMTMAP(R4)[R2],-    ; Setup map register data
                         1CCF  6143          CRB$L_INTD+VEC$W_MAPREG(R1) ;  and data path number
            37 A1    95  1CCF  6144         TSTB    CRB$L_INTD+VEC$B_DATAPATH(R1) ; Is the data path the DDP?
               12    13  1CD2  6145         BEQL    93$                       ; Br if yes - don't do purge or release
                         1CD4  6146         PUSHQ   R2                        ; Save R2, R3
                         1CD7  6147         PURDPR                            ; Purge the data path (ignore errors)
                         1CDD  6148         RELDPR                            ; Release the data path
                         1CE3  6149         POPQ    R2                        ; Restore R2, R3
               52    95  1CE6  6150 93$:    TSTB    R2                        ; Was it the pre-allocated one?
               0D    13  1CE8  6151         BEQL    95$                       ; Br if yes - clear data path number
    3C A442    01    CE  1CEA  6152         MNEGL   #1,CDB_L_XMTMAP(R4)[R2]    ; Indicate map register not allocated
                         1CEF  6153         RELMPR                            ; Release the map registers
               03    11  1CF5  6154         BRB     100$                      ; Complete the request
                         1CF7  6155
                         1CF7  6156 95$:    ASSUME  VEC$B_DATAPATH EQ VEC$W_MAPREG+3
            3F A4    94  1CF7  6157         CLRB    CDB_L_XMTMAP+3(R4)        ; Clear data path number used
       50   1A A3    3C  1CFA  6158 100$:   MOVZWL  CXB$W_BCNT(R3),R0         ; Get byte count of message
            51   50    D0  1CFE  6159         MOVL    R0,R1                     ; Copy converted length for accounting
       50   50   10    78  1D01  6160         ASHL    #16,R0,R0                 ; Move to high word
            50   01    B0  1D05  6161         MOVW    S^#SS$_NORMAL,R0          ; Set completion status
```

```
                39 05 A6    06    E1  1D08  6162          BBC     #XMT_FLG_V_ERRS,XMT_B_FLAGS(R6),110$ ; Br if not a FATAL error
                 50    0334 8F    B0  1D0D  6163          MOVW    #SS$_DEVREQERR,R0       ; Else, assume 16 retries failed
                45 06 A6    0A    E0  1D12  6164          BBS     #XMT_TERR_V_RTRY,XMT_W_TDRERR(R6),120$ ; Br if true
                 50    204C 8F    B0  1D17  6165          MOVW    #SS$_DISCONNECT,R0      ; Else, assume Loss of Carrier
                3B 06 A6    0B    E0  1D1C  6166          BBS     #XMT_TERR_V_LCAR,XMT_W_TDRERR(R6),120$ ; Br if true
                 50    005C 8F    B0  1D21  6167          MOVW    #SS$_DATACHECK,R0       ; Else, assume Late collision
                31 06 A6    0C    E0  1D26  6168          BBS     #XMT_TERR_V_LCOL,XMT_W_TDRERR(R6),120$ ; Br if true
                 50    20C4 8F    B0  1D2B  6169          MOVW    #SS$_COMMHARD,R0        ; Else, assume UNIBUS timeout
                27 06 A6    0E    E0  1D30  6170          BBS     #XMT_TERR_V_UBTO,XMT_W_TDRERR(R6),120$ ; Br if true
                 50    0054 8F    B0  1D35  6171          MOVW    #SS$_CTRLERR,R0         ; Else assume Data Underflow
                1D 06 A6    0D    E0  1D3A  6172          BBS     #XMT_TERR_V_UFLO,XMT_W_TDRERR(R6),120$ ; Br if true
                 50    029C 8F    B0  1D3F  6173          MOVW    #SS$_TOOMUCHDATA,R0     ; Else, must be Buffer Length violation
                          16    11  1D44  6174          BRB     120$                    ; Skip the accounting
                                    1D46  6175 ;
                                    1D46  6176 ; Perform accounting on a per protocol type basis.
                                    1D46  6177 ;
                                    1D46  6178 110$:    CNTR    R1,UCB$L_XE_SBYCTR(R5),L ; Bump the bytes sent counter
                                    1D52  6179          INCC    UCB$L_XE_SBUCTR(R5),L   ; Bump the blocks sent counter
                          24 A3     D5  1D5C  6180 120$:    TSTL    CXB$L_T_IRP(R3)         ; FAST interface CXB?
                             09     13  1D5F  6181          BEQL    130$                    ; Br if yes
                       53    24 A3  D0  1D61  6182          MOVL    CXB$L_T_IRP(R3),R3      ; Else, get IRP address
                           01E3     30  1D65  6183          BSBW    IO_DONE                 ; Finish the I/O request
                             03     11  1D68  6184          BRB     170$                    ; Continue in common code
                           00B8     30  1D6A  6185 130$:    BSBW    FINISH_XMT_FFI          ; Else, complete FFI XMIT
                           E738     30  1D6D  6186 170$:    BSBW    XMT_ALT_START           ; Start up any other transmits
                           FEF4     31  1D70  6187          BRW     60$                     ; Look for next completion
                                    1D73  6188
                              56 8ED0  1D73  6189 190$:    POPL    R6                      ; Restore R6
                                    05  1D76  6190          RSB
                                    1D77  6191
```

```
                        1D77  6193          .SBTTL   SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
                        1D77  6194 ;++
                        1D77  6195 ; Functional description:
                        1D77  6196 ;
                        1D77  6197 ; Subroutine to find SHR data structure for user
                        1D77  6198 ;
                        1D77  6199 ; Inputs:
                        1D77  6200 ;       R2 = Receive CXB address
                        1D77  6201 ;       R5 = UCB address
                        1D77  6202 ;
                        1D77  6203 ; Outputs:
                        1D77  6204 ;       R1 = Address if SHR data structure if match
                        1D77  6205 ;       All other registers preserved.
                        1D77  6206 ;       Z-Bit set then match.
                        1D77  6207 ;       Z-Bit clear then no match.
                        1D77  6208 ;--
                        1D77  6209
                        1D77  6210 MATCH_SRC:                                   ; Try to find shared user
              05   BB   1D77  6211          PUSHR   #^M<R0,R2>                  ; Save registers
                        1D79  6212 ;
                        1D79  6213 ; Try to find match among limited shared users of protocol type
                        1D79  6214 ;
        50  0098 C5   9E  1D79  6215          MOVAB   UCB$Q_XE_SHARE(R5),R0       ; Save address of listhead
        51   50  D0      1D7E  6216          MOVL    R0,R1                       ; Copy listhead address
                        1D81  6217          ASSUME  SHR_L_QFL EQ 0
        51   61  D0      1D81  6218 20$:     MOVL    (R1),R1                     ; Get next in list
        50   51  D1      1D84  6219          CMPL    R1,R0                       ; Back to start of list?
             06  13      1D87  6220          BEQL    30$                         ; Br if yes - no source match
             13  10      1D89  6221          BSBB    CHECK_SRC                   ; Check for match
             F4  12      1D8B  6222          BNEQ    20$                         ; Br if none
             0C  11      1D8D  6223          BRB     50$                         ; Return in success (Z-bit is set)
                        1D8F  6224 ;
                        1D8F  6225 ; No match on limited users - try to use default user
                        1D8F  6226 ;
    51  00C4 C5   D0      1D8F  6227 30$:     MOVL    UCB$L_XE_DEFUSR(R5),R1      ; Get address of default user
             02  13      1D94  6228          BEQL    40$                         ; Br if no default user
             50  D4      1D96  6229          CLRL    R0                          ; Return success
        50   50  D0      1D98  6230 40$:     MOVL    R0,R0                       ; Return success/failure indicator
             05  BA      1D9B  6231 50$:     POPR    #^M<R0,R2>                  ; Restore registers, don't reset Z-BIT
                 05      1D9D  6232          RSB
                        1D9E  6233
                        1D9E  6234 ;++
                        1D9E  6235 ; Functional description:
                        1D9E  6236 ;
                        1D9E  6237 ; Subroutine to check if source address in message matches SHR address
                        1D9E  6238 ;
                        1D9E  6239 ; Inputs:
                        1D9E  6240 ;       R1 = Address of SHR
                        1D9E  6241 ;       R2 = Address of MSG buffer
                        1D9E  6242 ;
                        1D9E  6243 ; Outputs:
                        1D9E  6244 ;       Z-Bit set then match.
                        1D9E  6245 ;       Z-Bit clear then no match.
                        1D9E  6246 ;--
                        1D9E  6247
                        1D9E  6248 CHECK_SRC:                                   ; Check for match with SHR data base
12  A1   3E  A2  D1      1D9E  6249          CMPL    CXB$G_R_SRC(R2),SHR_G_DEST(R1) ; Source address match?
```

```
                05   12  1DA3  6250          BNEQ    10$                         ; Br if no - try for next
            42  A2   B1  1DA5  6251          CMPW    CXB$G_R_SRC+4(R2),-         ; Really match?
            16  A1        1DA8  6252                   SHR_G_DEST+4(R1)           ;       ...
                05   1DAA  6253 10$:       RSB                                 ; Return to caller
```

```
                                1DAB  6255          .SBTTL  COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
                                1DAB  6256 ;++
                                1DAB  6257 ; COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
                                1DAB  6258 ;
                                1DAB  6259 ; Functional description:
                                1DAB  6260 ;
                                1DAB  6261 ; This routine allocates a receive buffer in which to copy the a receive
                                1DAB  6262 ; buffer for the PROMISCUOUS user.
                                1DAB  6263 ;
                                1DAB  6264 ;  Inputs:
                                1DAB  6265 ;
                                1DAB  6266 ;       R2 = Receive CXB address
                                1DAB  6267 ;       R4 = CDB address
                                1DAB  6268 ;       R5 = UCB address of PROMISCUOUS user
                                1DAB  6269 ;       R6 = Address of receive ring entry
                                1DAB  6270 ;
                                1DAB  6271 ;       IPL = FIPL
                                1DAB  6272 ;
                                1DAB  6273 ;  Outputs:
                                1DAB  6274 ;
                                1DAB  6275 ;       R0,R1,R3 are destroyed.
                                1DAB  6276 ;       All other registers are preserved.
                                1DAB  6277 ;--
                                1DAB  6278 COPY_RCV:                                    ; Copy the xmit buffer to rcv buffer
                     52   DD    1DAB  6279          PUSHL   R2                          ; Save address of original buffer
                1C A2     01   B1    1DAD  6280          CMPW    #1,CXB$W_R_NCHAIN(R2)   ; Is there more than 1 in chain?
                          6E   12    1DB1  6281          BNEQ    80$                     ; Br if yes - we can only handle 1
                   51  1A A2   3C    1DB3  6282          MOVZWL  CXB$W_BCNT(R2),R1       ; Get size of buffer
               51  00000048 8F   C0    1DB7  6283          ADDL    #CXB$C_HEADER,R1        ; Compute size of needed buffer
                   00000000'GF   16    1DBE  6284          JSB     G^EXE$ALONONPAGED       ; Allocate a receive buffer
                        5A 50    E9    1DC4  6285          BLBC    R0,80$                  ; Br if failure
                      53   6E    D0    1DC7  6286          MOVL    (SP),R3                 ; Get address of original buffer
                           36    BB    1DCA  6287          PUSHR   #^M<R1,R2,R4,R5>        ; Save registers
                62    63   51    28    1DCC  6288          MOVC3   R1,(R3),(R2)            ; Copy everything to new buffer
                           36    BA    1DD0  6289          POPR    #^M<R1,R2,R4,R5>        ; Save registers
                  08 A2    51    B0    1DD2  6290          MOVW    R1,CXB$W_SIZE(R2)       ; Reset size field
                                1DD6  6291 ;
                                1DD6  6292 ; If there is a pending receive I/O request, complete it.
                                1DD6  6293 ; Otherwise, queue the buffer and, if enabled, deliver attention AST.
                                1DD6  6294 ;
                   51    42 A5   3C    1DD6  6295          MOVZWL  UCB$W_DEVBUFSIZ(R5),R1  ; Get the user's buffer size
               53   00A8 D5    0F    1DDA  6296          REMQUE  @UCB$Q_XE_RCVREQ(R5),R3 ; Remove waiting IRP
                          0A   1D    1DDF  6297          BVS     30$                     ; Br if none - queue for later
               00C8 C5    51   A2    1DE1  6298          SUBW    R1,UCB$W_XE_QUOTA(R5)   ; Else, lessen quota so it can be
                                1DE6  6299                                              ;..increased on completion
                        0085    30    1DE6  6300          BSBW    FINISH_RCV_IO           ; And finish the I/O
                          36    11    1DE9  6301          BRB     80$                     ; Look for next completion
                                1DEB  6302
                                1DEB  6303 ;
                                1DEB  6304 ; Check buffer quota and queue if quota okay; otherwise, delete the buffer.
                                1DEB  6305 ;
                                1DEB  6306
               00C8 C5    51   A2    1DEB  6307 30$:    SUBW    R1,UCB$W_XE_QUOTA(R5)   ; Decrement the quota
                          24   1E    1DF0  6308          BGEQU   50$                     ; Br if we can buffer request
               00C8 C5    51   A0    1DF2  6309          ADDW    R1,UCB$W_XE_QUOTA(R5)   ; Replace quota
                                1DF7  6310          INCC    UCB$W_XE_UBOCTR(R5),W   ; Else, no buffer available
                                1E01  6311          INCC    CDB_W_UBOCTR(R4),W      ; ...don't forget CDB counter
```

```
                50    52    D0  1E0B  6312          MOVL    R2,R0                           ; Copy the buffer address
 00000000'GF        16    1E0E  6313          JSB     G^COM$DRVDEALMEM                ; DEALLOCATE the buffer
          0B        11    1E14  6314          BRB     80$                             ; Done
                          1E16  6315
 00A4 D5    62        0E  1E16  6316 50$:     INSQUE  (R2),@UCB$Q_XE_RCVMSG+4(R5) ; Queue received msg for later
        F58F        30    1E1B  6317          BSBW    FILLRCVLIST                     ; Try to fill the receive list
        12A9        30    1E1E  6318          BSBW    POKE_USER                       ; Deliver ASTs
          52  8ED0   1E21  6319 80$:     POPL    R2                              ; Restore R2
              05    1E24  6320          RSB                                     ; Return to caller
```

```
                     1E25  6322          .SBTTL  FINISH_XMT_FFI - Finish FAST interface transmit processing
                     1E25  6323 ;++
                     1E25  6324 ; FINISH_XMT_FFI - Finish FAST interface transmit processing
                     1E25  6325 ;
                     1E25  6326 ; Functional description:
                     1E25  6327 ;
                     1E25  6328 ; This routine completes a transmit CXB for a particular user of the fast
                     1E25  6329 ; interface.
                     1E25  6330 ;
                     1E25  6331 ;  Inputs:
                     1E25  6332 ;
                     1E25  6333 ;       R0 = Status of transmit request
                     1E25  6334 ;       R3 = transmit CXB address
                     1E25  6335 ;       R4 = CDB address
                     1E25  6336 ;       R5 = UCB address
                     1E25  6337 ;
                     1E25  6338 ;       IPL = FIPL
                     1E25  6339 ;
                     1E25  6340 ;  Outputs:
                     1E25  6341 ;
                     1E25  6342 ;       R0-R3 are destroyed.
                     1E25  6343 ;       All other registers are preserved.
                     1E25  6344 ;
                     1E25  6345 ;--
                     1E25  6346          ASSUME  IPL$_SYNCH EQ IPL$_XE_FIPL
                     1E25  6347 FINISH_XMT_FFI::                             ; Finish FAST interface transmit request
           54    DD  1E25  6348          PUSHL   R4                          ; Save R4
54   0178 C5    D0  1E27  6349          MOVL    UCB$L_XE_FFI(R5),R4         ; Get FFI block address
     14 B4    16  1E2C  6350          JSB     @FFI$L_XMIT_DONE(R4)        ; Call back the user with CXB
        54 8ED0  1E2F  6351          POPL    R4                          ; Restore R4
           05  1E32  6352          RSB
```

```
                        1E33  6354          .SBTTL  FINISH_RCV_FFI - Finish FAST receive processing
                        1E33  6355 ;++
                        1E33  6356 ; FINISH_RCV_FFI - Finish FAST receive processing
                        1E33  6357 ;
                        1E33  6358 ; Functional description:
                        1E33  6359 ;
                        1E33  6360 ; This routine completes a receive CXB for a particular user of the fast
                        1E33  6361 ; interface.
                        1E33  6362 ;
                        1E33  6363 ;  Inputs:
                        1E33  6364 ;
                        1E33  6365 ;       R2 = receive CXB address
                        1F33  6366 ;       R4 = CDB address
                        1E33  6367 ;       R5 = UCB address
                        1E33  6368 ;
                        1E33  6369 ;       IPL = FIPL
                        1E33  6370 ;
                        1E33  6371 ;  Outputs:
                        1E33  6372 ;
                        1E33  6373 ;       R0-R3 are destroyed.
                        1E33  6374 ;       All other registers are preserved.
                        1E33  6375 ;
                        1E33  6376 ;--
                        1E33  6377          ASSUME  IPL$_SYNCH EQ IPL$_XE_FIPL
                        1E33  6378 FINISH_RCV_FFI::                         ; Finish FAST recieve request
          54    DD      1E33  6379          PUSHL   R4                      ; Save CDB
     53   52    D0      1E35  6380          MOVL    R2,R3                   ; Copy CXB address
       3E A3    7D      1E38  6381          MOVQ    CXB$G_R_SRC(R3),-       ; Copy source address
       28 A3           1E3B  6382                  CXB$Q_STATION(R3)       ;      ...
 18 A3 0046 8F    B0    1E3D  6383          MOVW    #CXB$T_R_USERDAT,CXB$W_BOFF(R3) ; Set offset to received data
                        1E43  6384          ASSUME  NMA$C_STATE_ON EQ 0
                        1E43  6385          ASSUME  NMA$C_STATE_OFF EQ 1
    09 00D9 C5    E8    1E43  6386          BLBS    UCB$B_XE_PAD(R5),30$    ; Br if padding is disabled
       46 A3    B0      1E48  6387          MOVW    CXB$W_R_SIZE(R3),-      ; Else, set real size of buffer
       1A A3           1E4B  6388                  CXB$W_BCNT(R3)
    18 A3    02    A0    1E4D  6389          ADDW    #XE_C_CNTSIZ,CXB$W_BOFF(R3) ; Adjust offset
 50 00C4 C4    D0      1E51  6390 30$:      MOVL    CDB_L_DEVDEPEND(R4),R0  ; Set controller bits
    50   44 A5    C8    1E56  6391          BISL    UCB$L_DEVDEPEND(R5),R0  ; Set status flags
 54 0178 C5    D0      1E5A  6392          MOVL    UCB$L_XE_FFI(R5),R4     ; Get FFI block address
       18 B4    16      1E5F  6393          JSB     @FFI$L_RECV_DONE(R4)    ; Call back the user with CXB
          54 8ED0       1E62  6394          POPL    R4                      ; Restore CDB
    52   53    D0      1E65  6395          MOVL    R3,R2                   ; Was buffer consumed?
          03    13      1E68  6396          BEQL    90$                     ; Br if YES
       F542    30      1E6A  6397          BSBW    ADDRCVLIST              ; Else, add buffer to receive list
               05      1E6D  6398 90$:      RSB
```

```
                                1E6E  6400          .SBTTL  FINISH_RCV_IO - Finish receive I/O processing
                                1E6E  6401 ;++
                                1E6E  6402 ; FINISH_RCV_IO - Finish receive I/O processing
                                1E6E  6403 ;
                                1E6E  6404 ; Functional description:
                                1E6E  6405 ;
                                1E6E  6406 ; This routine completes a receive operation that has been matched with a
                                1E6E  6407 ; message block. After the receive has been completed the message free list
                                1E6E  6408 ; is filled and a receive is started if needed.
                                1E6E  6409 ;
                                1E6E  6410 ;  Inputs:
                                1E6E  6411 ;
                                1E6E  6412 ;       R2 = receive CXB address
                                1E6E  6413 ;       R3 = I/O packet address
                                1E6E  6414 ;       R4 = CDB address
                                1E6E  6415 ;       R5 = UCB address
                                1E6E  6416 ;
                                1E6E  6417 ;       IPL = FIPL
                                1E6E  6418 ;
                                1E6E  6419 ;  Outputs:
                                1E6E  6420 ;
                                1E6E  6421 ;       R5 is reset to UCB address from IRP
                                1E6E  6422 ;
                                1E6E  6423 ;       The request is completed via I/O post.
                                1E6E  6424 ;--
                                1E6E  6425
                                1E6E  6426 FINISH_RCV_IO::                          ; Finish recieve I/O request
          2C A3    52    D0    1E6E  6427          MOVL    R2,IRP$L_SVAPTE(R3)       ; Save block address
               3E A2    7D    1E72  6428          MOVQ    CXB$G_R_SRC(R2),-         ; Copy source address for DECnet
                  40 A3          1E75  6429                  IRP$Q_STATION(R3)         ;       ...
                     07    E1    1E77  6430          BBC     #IRP$V_DIAGBUF,-          ; Br if no diagnotic buffer
                        27 2A A3       1E79  6431                  IRP$W_STS(R3),10$         ;
                  51    4C A3    D0    1E7C  6432          MOVL    IRP$L_DIAGBUF(R3),R1      ; Get diagnostic buffer
                  50    0C A1    9E    1E80  6433          MOVAB   RHDR_T_DATA(R1),R0        ; Assume this is just a read header
                  08 A1    1A    B1    1E84  6434          CMPW    #RHDR_C_LENGTH,DIAG_W_SIZE(R1) ; Is this just a header buffer?
                           10    13    1E88  6435          BEQL    5$                        ; Br if yes
                  50    3E A1    9E    1E8A  6436          MOVAB   DIAG_T_RDATA(R1),R0       ; Else, must be a diagnostic buffer
                     0B A2    90    1E8E  6437          MOVB    CXB$B_R_FLAGS(R2),-       ; Save diagnostic return info
                        00CD C4       1E91  6438                   CDB_B_DIAG1(R4)          ;       ...
                     14 A2    B0    1E94  6439          MOVW    CXB$W_R_LENERR(R2),-      ;       ...
                        00CE C4       1E97  6440                   CDB_W_DIAG2(R4)          ;       ...
                           3C    BB    1E9A  6441 5$:      PUSHR   #^M<R2,R3,R4,R5>          ; Save registers
          60    38 A2    0E    28    1E9C  6442          MOVC3   #RHDR_C_DATA,CXB$T_R_DATA(R2),(R0) ; Move header info
                           3C    BA    1EA1  6443          POPR    #^M<R2,R3,R4,R5>          ; Restore registers
                  62    46 A2    9E    1EA3  6444 10$:     MOVAB   CXB$T_R_USERDAT(R2),(R2); Set address of received data
             04 A2    3C A3    D0    1EA7  6445          MOVL    IRP$L_XE_DATBUF(R3),4(R2); Set address of user buffer
                     42 A5    A0    1EAC  6446          ADDW    UCB$W_DEVBUFSIZ(R5),-     ; Adjust receive buffer quota
                        00C8 C5       1EAF  6447                  UCB$W_XE_QUOTA(R5)        ;
                  51    1A A2    3C    1EB2  6448          MOVZWL  CXB$W_BCNT(R2),R1         ; Find length of received message
                                       1EB6  6449                                            ;  includes the CRC size
                                       1EB6  6450 ;
                                       1EB6  6451 ; Perform accounting on a per protocol type basis.
                                       1EB6  6452 ;
                                       1EB6  6453          CNTR    R1,UCB$L_XE_RBYCTR(R5),L ; Bump the bytes received counter
                                       1EC2  6454          INCC    UCB$L_XE_RBCCTR(R5),L    ; Bump the blocks received counter
                              51    0E    C2    1ECC  6455          SUBL    #XE_C_HEADER,R1          ; Remove DEUNA/DELUA header from size
                                       1ECF  6456 ;
```

```
                                1ECF  6457 ; If padding is enabled, then the size of the data is contained in the
                                1ECF  6458 ; message as the first word of data.
                                1ECF  6459 ;
                                1ECF  6460         ASSUME  NMA$C_STATE_ON EQ 0
                                1ECF  6461         ASSUME  NMA$C_STATE_OFF EQ 1
          10 00D9 C5    E8      1ECF  6462         BLBS    UCB$B_XE_PAD(R5),15$    ; Br if padding is disabled
    50    46 A2   04    A1      1ED4  6463         ADDW3   #XE_C_CRC,CXB$W_R_SIZE(R2),R0 ; Else, pick up real size of
                                1ED9  6464                                         ;  MSG from the message itself + CRC
             62   02    C0      1ED9  6465         ADDL    #XE_C_CNTSIZ,(R2)       ; Move pointer past the count field
                                1EDC  6466 ;
                                1EDC  6467 ; Verify that the 1st word of data at least makes some sense. The byte
                                1EDC  6468 ; count for the message must be less than the size of the entire received
                                1EDC  6469 ; message.
                                1EDC  6470 ;
             51   50    B1      1EDC  6471         CMPW    R0,R1                   ; Is size field larger than buffer?
                  44    1E      1EDF  6472         BGEQU   45$                     ; Br if yes -
                                1EE1  6473                                         ;  must be strictly Less Than, because
                                1EE1  6474                                         ;  the size field is 2 extra bytes.
             51   50    3C      1EE1  6475         MOVZWL  R0,R1                   ; Else, copy the real buffer size
                                1EE4  6476 ;
                                1EE4  6477 ; We will attempt to return the CRC as part of the user data, but not
                                1EE4  6478 ; tell the user that the CRC was returned (i.e. we do not set IOST1 size
                                1EE4  6479 ; to include the CRC). If the CRC cannot be returned, then we only return
                                1EE4  6480 ; the data portion of the buffer.
                                1EE4  6481 ;
                                1EE4  6482 ; This is done, so that users suspecting that data is being "picked" between
                                1EE4  6483 ; the UBA and host system memory can check the CRC for themselves.
                                1EE4  6484 ;
          3A A3   01    9B      1EE4  6485 15$:    MOVZBW  S^#SS$_NORMAL,IRP$W_XE_STATUS(R3) ; Assume success
                  50    04    9A      1EE8  6486         MOVZBL  #XE_C_CRC,R0            ; Get size of CRC
          32 A3   51    B1      1EEB  6487         CMPW    R1,IRP$W_BCNT(R3)       ; Request larger than user buffer size?
                  17    1B      1EEF  6488         BLEQU   20$                     ; Br if no - okay
             51   50    C2      1EF1  6489         SUBL    R0,R1                   ; Try it without the CRC being returned
                  2F    1B      1EF4  6490         BLEQU   45$                     ; Br if DEUNA/DELUA hardware error
                  50    D4      1EF6  6491         CLRL    R0                      ; Remember that CRC is not returned
             32 A3   51    B1      1EF8  6492         CMPW    R1,IRP$W_BCNT(R3)       ; Request larger than user buffer size?
                  0A    1B      1EFC  6493         BLEQU   20$                     ; Br if no - okay (without CRC)
          51    32 A3   3C      1EFE  6494         MOVZWL  IRP$W_BCNT(R3),R1       ; Else, set size to minimum of two
    3A A3    0838 8F    B0      1F02  6495         MOVW    #SS$_DATAOVERUN,IRP$W_XE_STATUS(R3) ; Set return status
                                1F08  6496 ;
                                1F08  6497 ; If chained buffers, then setup IRP$W_STS and don't reset the USER BUFFER
                                1F08  6498 ; SIZE (IRP$W_BCNT), because chained buffers need to have the USER BUFFER
                                1F08  6499 ; SIZE in IRP$W_BCNT.
                                1F08  6500 ;
            10 A2   D5      1F08  6501 20$:    TSTL    CXB$L_LINK(R2)          ; Is this a complex chained buffer?
                  0B    13      1F0B  6502         BEQL    30$                     ; Br if no - set transfer size
                  05    E1      1F0D  6503         BBC     #IRP$V_CHAINED,-        ; Br if user cannot accept complex
             13 2A A3             1F0F  6504                 IRP$W_STS(R3),45$       ;  chained buffers
                  08    A8      1F12  6505         BISW    #IRP$M_COMPLX,-         ; Else indicate complex
               2A A3             1F14  6506                 IRP$W_STS(R3)           ;    chained buffers
                  04    11      1F16  6507         BRB     40$                     ; And don't change user buffer size
          32 A3   51    B0      1F18  6508 30$:    MOVW    R1,IRP$W_BCNT(R3)       ; Set size to transfer
             51   50    C2      1F1C  6509 40$:    SUBL    R0,R1                   ; Return receive count without CRC
    50       51   10    78      1F1F  6510         ASHL    #16,R1,R0               ; Set buffer size in status
                  06    12      1F23  6511         BNEQ    50$                     ; Br if success
               0054 8F    B0      1F25  6512 45$:    MOVW    #SS$_CTRLERR,-          ; Set data transfer error
                  3A A3             1F29  6513                 IRP$W_XE_STATUS(R3)     ;
```

```
                                1F2B  6514
          50    3A A3    B0     1F2B  6515 50$:     MOVW    IRP$W_XE_STATUS(R3),R0  ; Get status
                   1A    10     1F2F  6516 60$:     BSBB    IO_DONE                 ; Post the I/O request
                 F479    31     1F31  6517          BRW     FICLRCVLIST             ; Fill up the receive buffers
                                1F34  6518 ;
                                1F34  6519 ; Complete an I/O request packet
                                1F34  6520 ;
                                1F34  6521 ABORT_PKT:                               ; Abort the I/O request
             50    2C    9A     1F34  6522          MOVZBL  S^#SS$_ABORT,R0         ; Return aborted status
                                1F37  6523 IO_DONE3:                                ; Complete the I/O request, check for
                                1F37  6524 ;                                        ;    timeout
          55    1C A3    D0     1F37  6525          MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
        51    0180 C5    D0     1F3B  6526          MOVL    UCB$L_XE_CDB(R5),R1     ; Get CDB address
    05 00C4 C1    09     E1     1F40  6527          BBC     #XM$V_STS_TIMO,CDB_L_DEVDEPEND(R1),IO_DONE ; Br if not a timeout
        50    022C 8F    3C     1F46  6528          MOVZWL  #SS$_TIMEOUT,R0         ; Else, return real error code
                                1F4B  6529 IO_DONE:                                 ; Complete an I/O request
          38 A3    50    D0     1F4B  6530          MOVL    R0,IRP$L_IOST1(R3)      ; Set status return and size
                                1F4F  6531 IO_DONE1:                                ; Alternate entry point
          55    1C A3    D0     1F4F  6532          MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
       3C A3    44 A5    D0     1F53  6533          MOVL    UCB$L_DEVDEPEND(R5),IRP$L_IOST2(R3) ; Set other info
        51    0180 C5    D0     1F58  6534          MOVL    UCB$L_XE_CDB(R5),R1     ; Get CDB address
                   06    13     1F5D  6535          BEQL    IO_DONE2                ; Br if no CDB
    3C A3    00C4 C1    C8     1F5F  6536          BISL    CDB_L_DEVDEPEND(R1),IRP$L_IOST2(R3) ; Set controller bits
                                1F65  6537 IO_DONE2:                                ; P1 IOST2 already setup
                   54    DD     1F65  6538          PUSHL   R4                      ; Save R4
        54    0180 C5    D0     1F67  6539          MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
                   07    E1     1F6C  6540          BBC     #IRP$V_DIAGBUF,-        ; Br if no diagnotic buffer
             22 2A A3          1F6E  6541                  IRP$W_STS(R3),20$       ;
          50    4C A3    D0     1F71  6542          MOVL    IRP$L_DIAGBUF(R3),R0    ; Get diagnostic buffer address
          08 A0    1A    B1     1F75  6543          CMPW    #RHDR_C_LENGTH,DIAG_W_SIZE(R0) ; Is this a real diag buffer?
                   12    13     1F79  6544          BEQL    20$                     ; Br if not - no diagnostic info
        50    60    08    C1     1F7B  6545          ADDL3   #8,(R0),R0              ; Address buffer past start time
    80   00000000'GF    7D     1F7F  6546          MOVQ    G^EXE$GQ_SYSTIME,(R0)+  ; Insert stop time
        51    00C8 C4    D0     1F86  6547          MOVL    CDB_L_UCB0(R4),R1       ; Get address of UCB #0
        80    0082 C1    3C     1F8B  6548          MOVZWL  UCB$W_ERRCNT(R1),(R0)+  ; Insert error counter
                 0263    30     1F90  6549          BSBW    REG_DUMP                ; Dump registers
                   54 8ED0    1F93  6550 20$:     POPL    R4                      ; Restore R4
         00000000'GF    17     1F96  6551          JMP     G^COM$POST              ; Post the I/O and return
```

```
                        1F9C  6553          .SBTTL  ASSPKT_NODCH - Assemble receive packets without Data Chaining
                        1F9C  6554 ;++
                        1F9C  6555 ; ASSPKT_NODCH - Assemble receive packets With Data Chaining Disabled
                        1F9C  6556 ;
                        1F9C  6557 ; Functional description:
                        1F9C  6558 ;
                        1F9C  6559 ; This routine finds and returns the first recieve packet that the controller
                        1F9C  6560 ; has completed. This routine is invoked to assemble recieve packets if and
                        1F9C  6561 ; only if data chaining has been disabled.
                        1F9C  6562 ;
                        1F9C  6563 ;  Inputs:
                        1F9C  6564 ;
                        1F9C  6565 ;       R4 = CDB address
                        1F9C  6566 ;
                        1F9C  6567 ;       IPL = FIPL
                        1F9C  6568 ;
                        1F9C  6569 ;  Outputs:
                        1F9C  6570 ;
                        1F9C  6571 ;       R0 = Status for request
                        1F9C  6572 ;       R2 = Address of receive buffer
                        1F9C  6573 ;       R6 = Address of corresponding receive ring descriptor entry
                        1F9C  6574 ;       R1,R3 are destroyed.
                        1F9C  6575 ;       All other registers are preserved.
                        1F9C  6576 ;
                        1F9C  6577 ;--
                        1F9C  6578
                        1F9C  6579 ASSPKT_NODCH::                               ; Assemble receive packet
 0105    30             1F9C  6580          BSBW    NEXTMSG                     ; Get message
01 50    E8             1F9F  6581          BLBS    R0,5$                       ; Branch if one
         05             1FA2  6582          RSB                                 ; Else return in error
                        1FA3  6583 ;
                        1FA3  6584 ; Set CXB$W_R_NCHAIN, the number of messages in the chain, to 1 and
                        1FA3  6585 ; place the size of the recieved message in CXB$W_BCNT.
                        1FA3  6586 ;
                        1FA3  6587
1C A2    01             1FA3  6588 5$:      MOVW    S^#1,CXB$W_R_NCHAIN(R2) ; There is only one buffer
   0C A2    B0          1FA7  6589          MOVW    CXB$W_LENGTH(R2),-      ; Set the message size
   1A A2                1FAA  6590                  CXB$W_BCNT(R2)
                        1FAC  6591
                        1FAC  6592 ;
                        1FAC  6593 ; If the current message is both the beginning and end of a packet, then this
                        1FAC  6594 ; message will be returned (if there was no error in its reception). Otherwise,
                        1FAC  6595 ; return the buffer to the free list, and attempt to get the next message.
                        1FAC  6596 ;
                        1FAC  6597
                        1FAC  6598          ASSUME  RCV_FLG_V_ENP EQ 0
                        1FAC  6599          ASSUME  RCV_FLG_V_STP EQ RCV_FLG_V_ENP+1
   02    00    ED       1FAC  6600          CMPZV   #RCV_FLG_V_ENP,#2,-      ; Is this message both the beginning
03    0B A2              1FAF  6601                  CXB$B_R_FLAGS(R2),#3    ; and the end of a packet?
         74    13       1FB2  6602          BEQLU   CRC                         ; Branch if so
                        1FB4  6603          INCC    CDB_W_UFDCTR(R4),W          ; Else increment counter
      F3EE    30        1FBE  6604          BSBW    ADDRCVLIST                  ; Add the buffer to the recieve list
         D9    11       1FC1  6605          BRB     ASSPKT_NODCH                ; Continue to try and get the next msg
```

```
                     1FC3  6607          .SBTTL  ASSPKT_DCH - Assemble receive packets with Data Chaining Enabled
                     1FC3  6608 ;++
                     1FC3  6609 ; ASSPKT_DCH - Assemble receive packets with Data Chaining Enabled
                     1FC3  6610 ;
                     1FC3  6611 ; Functional description:
                     1FC3  6612 ;
                     1FC3  6613 ; This routine assembles all receive packets into one chain of complex
                     1FC3  6614 ; buffers. This routine is invoked to assemble recieve packets if and only if
                     1FC3  6615 ; data chaining has been enabled.
                     1FC3  6616 ;
                     1FC3  6617 ;  Inputs:
                     1FC3  6618 ;
                     1FC3  6619 ;       R4 = CDB address
                     1FC3  6620 ;
                     1FC3  6621 ;       IPL = FIPL
                     1FC3  6622 ;
                     1FC3  6623 ;  Outputs:
                     1FC3  6624 ;
                     1FC3  6625 ;       R0 = Status for request
                     1FC3  6626 ;       R2 = Address of first receive buffer in chain
                     1FC3  6627 ;       R6 = Address of last buffer in receive ring
                     1FC3  6628 ;       R1,R3 are destroyed.
                     1FC3  6629 ;       All other registers are preserved.
                     1FC3  6630 ;
                     1FC3  6631 ;  Implicit Outputs:
                     1FC3  6632 ;
                     1FC3  6633 ;       IRP$V_CHAIN and IRP$V_COMPLX bits set in IRP$W_STS if the receive
                     1FC3  6634 ;       buffer is comprised of complex chained buffers.
                     1FC3  6635 ;--
                     1FC3  6636          .ENABL  LSB
                     1FC3  6637 ASSPKT_DCH::                            ; Assemble receive packets
        00DE     30  1FC3  6638          BSBW    NEXTMSG                 ; Get first message
      01 50      E8  1FC6  6639          BLBS    R0,5$                   ; Br if one
                 05  1FC9  6640          RSB                             ; Return in error
                     1FCA  6641 ;
                     1FCA  6642 ; Save number of messages  in chain in CXB$W_R_NCHAIN and total size of
                     1FCA  6643 ; all messages in CXB$W_BCNT
                     1FCA  6644 ;
  1C A2    01    B0  1FCA  6645 5$:     MOVW    S^#1,CXB$W_R_NCHAIN(R2) ; Compute total number of buffers
     1A A2    B4  1FCE  6646          CLRW    CXB$W_BCNT(R2)          ; Init total size of buffers so far
                     1FD1  6647 ;
                     1FD1  6648 ; Toss message if not first in chain.
                     1FD1  6649 ;
        01    E0  1FD1  6650          BBS     #RCV_FLG_V_STP,-        ; Br if beginning of packet
  03 0B A2                1FD3  6651                  CXB$B_R_FLAGS(R2),7$ ;
        00A5    31  1FD6  6652          BRW     TOSSMSG                 ; Else, toss the packet
                     1FD9  6653
        00    E0  1FD9  6654 7$:     BBS     #RCV_FLG_V_ENP,-        ; Br if end of packet
  42 0B A2                1FDB  6655                  CXB$B_R_FLAGS(R2),20$ ; ..all done with this loop
        52    DD  1FDE  6656          PUSHL   R2                      ; Save first receive buffer address
   53   01    9A  1FE0  6657          MOVZBL  #MAX_C_CHAIN,R3         ; Allow n messages in chain
        66    B0  1FE3  6658 10$:    MOVW    RCV_Q_SLEN(R6),-        ; Set size of buffer to maximum per rcv
     0C A2              1FE5  6659                  CXB$W_LENGTH(R2)    ;
        52    DD  1FE7  6660          PUSHL   R2                      ; Save address of current bufr in chain
      00B8    30  1FE9  6661          BSBW    NEXTMSG                 ; Try for next message
        51 8ED0   1FEC  6662          POPL    R1                      ; Get address of last buffer in chain
     17 50    E9  1FEF  6663          BLBC    R0,12$                  ; Toss all messages on error
```

```
                50    6E    D0  1FF2  6664          MOVL    (SP),R0                 ; Get address of first in chain
            10  A1    52    D0  1FF5  6665          MOVL    R2,CXB$L_LINK(R1)       ; Store address in chain
                0C    A1    A0  1FF9  6666          ADDW    CXB$W_LENGTH(R1),-      ; Compute total size of all buffers
                1A    A0        1FFC  6667           CXB$W_BCNT(R0)         ;  in chain - so far
                1C    A0    B6  1FFE  6668          INCW    CXB$W_R_NCHAIN(R0)      ; Compute number of msgs in chain
                      00    E0  2001  6669          BBS     #RCV_FLG_V_ENP,-        ; Br if end of packet
             05 0B    A2        2003  6670           CXB$B_R_FLAGS(R2),15$   ;
                DA    53    F5  2006  6671          SOBGTR  R3,10$                  ; Loop if more than two in chain
                            2009  6672 ;
                            2009  6673 ; Done with loop and ENP bit still not set - toss all messages
                            2009  6674 ;
                      70    11  2009  6675 12$:     BRB     TOSAMSG                 ; Toss all messages
                            200B  6676
                51    52    D0  200B  6677 15$:     MOVL    R2,R1                   ; Save address of last message
                      52  8ED0  200E  6678          POPL    R2                      ; Return first message address in R2
                   0B    A1    90  2011  6679          MOVB    CXB$B_R_FLAGS(R1),-     ; Save only last message buffer info
                   0B    A2        2014  6680           CXB$B_R_FLAGS(R2)      ;  in first message of chain
                   14    A1    B0  2016  6681          MOVW    CXB$W_R_LENERR(R1),-    ; DITTO
                   14    A2        2019  6682           CXB$W_R_LENERR(R2)     ;
                   1A    A2    A2  201B  6683          SUBW    CXB$W_BCNT(R2),-        ; Compute size of last message
                   0C    A1        201E  6684           CXB$W_LENGTH(R1)       ;  and store in CXB format
                F000  8F    AB  2020  6685 20$:     BICW3   #^C<RCV_LERR_M_MLEN>,-  ; Return size of complete message
                   14    A2        2024  6686           CXB$W_R_LENERR(R2),-    ;
                   1A    A2        2026  6687           CXB$W_BCNT(R2)         ;
                            2028  6688 ;
                            2028  6689 ; We will only check CRCs on messages looped back to ourselves.
                            2028  6690 ;
                      03    E1  2028  6691 CRC::    BBC     #RCV_FLG_V_CRC,-        ; Br if not a CRC error
             4A 0B    A2        202A  6692           CXB$B_R_FLAGS(R2),50$   ;                    ...
                   3E    A2    D1  202D  6693          CMPL    CXB$G_R_SRC(R2),-       ; Is this a "looped" message?
                026F  C4        2030  6694           CDB_G_PHYADR(R4)       ;
                      42    12  2033  6695          BNEQ    50$                     ; Br if not - drop it
                   42    A2    B1  2035  6696          CMPW    CXB$G_R_SRC+4(R2),-     ; Is it really "looped"?
                0273  C4        2038  6697           CDB_G_PHYADR+4(R4)     ;
                      3A    12  203B  6698          BNEQ    50$                     ; Br if not - drop it
                            203D  6699 ;
                            203D  6700 ; There is not need to check the source address of the message, since
                            203D  6701 ; while we are in loopback mode, we cannot receive messages from other
                            203D  6702 ; sources.
                            203D  6703 ;
                            203D  6704 ; Calculate CRC on message
                            203D  6705 ;
                   50    01    CE  203D  6706          MNEGL   #1,R0                   ; Initialize CRC value
                   53    52    D0  2040  6707          MOVL    R2,R3                   ; Copy address of first in chain
                         04    11  2043  6708          BRB     35$                     ; Skip first chain
                53    10    A3    D0  2045  6709 30$:     MOVL    CXB$L_LINK(R3),R3       ; Get address of next in chain
                51    0C    A3    3C  2049  6710 35$:     MOVZWL  CXB$W_LENGTH(R3),R1     ; Get size of data
                   10    A3    D5  204D  6711          TSTL    CXB$L_LINK(R3)          ; Is this the end of the data?
                         03    12  2050  6712          BNEQ    40$                     ; Br if no
                   51    04    C2  2052  6713          SUBL    #XE_C_CRC,R1            ; Subtract out the CRC
                         0E    BB  2055  6714 40$:     PUSHR   #^M<R1,R2,R3>           ; Save registers
38 A3    51    50    E166  CF    0B  2057  6715          CRC     CRCTABLE,R0,R1,CXB$T_R_DATA(R3) ; Calculate CRC on data
                         0E    BA  205F  6716          POPR    #^M<R1,R2,R3>           ; Restore registers
                   10    A3    D5  2061  6717          TSTL    CXB$L_LINK(R3)          ; Is there a next?
                         DF    12  2064  6718          BNEQ    30$                     ; Br if YES
                   51    53    C0  2066  6719          ADDL    R3,R1                   ; Point to end of buffer
                   50    50    D2  2069  6720          MCOML   R0,R0                   ; Complement the CRC
```

```
                38 A1    50    D1  206C  6721          CMPL    R0,CXB$T_R_DATA(R1)     ; Is the CRC good?
                   05    12  2070  6722          BNEQ    50$                     ; Br if not
                   48 8F 8A  2072  6723          BICB    #RCV_FLG_M_ERRS!RCV_FLG_M_CRC,- ; Else, indicate good data
                   0B A2     2075  6724                  CXB$B_R_FLAGS(R2)       ;
                50    01    9A  2077  6725 50$:     MOVZBL  S^#SS$_NORMAL,R0        ; Return success!
                         05  207A  6726 100$:    RSB
                             207B  6727 ;
                             207B  6728 ; Toss bad messages
                             207B  6729 ;
                      52 8ED0  207B  6730 TOSAMSG:POPL    R2                      ; Restore R2
                             207E  6731 TOSSMSG:INCC    CDB_W_UFDCTR(R4),W      ; Up the counter
                   10 A2    DD  2088  6732 110$:    PUSHL   CXB$L_LINK(R2)          ; Save address of next in chain
                    F321    30  208B  6733          BSBW    ADDRCVLIST              ; Add buffer to receive list
                      52 8ED0  208E  6734          POPL    R2                      ; Restore address of next in chain
                      F5    12  2091  6735          BNEQ    110$                    ; Br if more in chain
                      50    D4  2093  6736 120$:    CLRL    R0                      ; Assume failure
                      00    E1  2095  6737          BBC     #RCV_FLG_V_ENP,-        ; Br if NOT end of chain
                03 05 A6        2097  6738                   RCV_B_FLAGS(R6),130$   ;  get rest of message
                    FF26    31  209A  6739          BRW     ASSPKT_DCH              ; Else, try for next valid message
                      05    10  209D  6740 130$:    BSBB    NEXTMSG                 ; Get next message
                   D8 50    E9  209F  6741          BLBC    R0,100$                 ; Br if none
                      EF    11  20A2  6742          BRB     120$                    ; Check if more possible
                                20A4  6743          .DSABL  LSB
                                20A4  6744 ;
                                20A4  6745 ; Find next message and check own bit
                                20A4  6746 ;
                                20A4  6747 NEXTMSG::
                      50    D4  20A4  6748          CLRL    R0                      ; Assume failure
                   00B6 C4    95  20A6  6749          TSTB    CDB_B_RCVCNT(R4)        ; Any more receives in progress?
                         63    13  20AA  6750          BEQL    20$                     ; Br if not
                56    00B5 C4    9A  20AC  6751          MOVZBL  CDB_B_LASTRCV(R4),R6    ; Get last ring entry completed
                56    5C A446    D0  20B1  6752          MOVL    CDB_L_RRADDR(R4)[R6],R6 ; Retrieve last ring entry address
             54 05 A6       07    E0  20B6  6753          BBS     #RCV_FLG_V_OWN,RCV_B_FLAGS(R6),20$ ; Br if not done
                52    00A4 D4    0F  20BB  6754          REMQUE  @CDB_Q_RCVPND(R4),R2    ; Get next receive
                         4D    1D  20C0  6755          BVS     20$                     ; Br if none available (yet)
                   10 A2    D4  20C2  6756          CLRL    CXB$L_LINK(R2)          ; Assume not a chained buffer
                   00B6 C4    97  20C5  6757          DECB    CDB_B_RCVCNT(R4)        ; One less receive pending
                   00B5 C4    96  20C9  6758          INCB    CDB_B_LASTRCV(R4)       ; Bump ring pointer
                     F8 8F    8A  20CD  6759          BICB    #^C<MAX_C_RCV-1>,-      ; Modulo receive ring entry size
                   00B5 C4        20D0  6760                   CDB_B_LASTRCV(R4)      ;       ...
                                20D3  6761
             22 A2    08 A6    B1  20D3  6762          CMPW    RCV_W_RID(R6),CXB$W_XE_RID(R2) ; Is this a match???
                         04    13  20D8  6763          BEQL    10$                     ; Br if yes, good
                                20DA  6764          BUG_CHECK NOBUFPCKT,FATAL       ; Else, fatal error
                                20DE  6765 ;
                                20DE  6766 ; Adjust quota and release mapping slot
                                20DE  6767 ;
                   06 A6    B0  20DE  6768 10$:     MOVW    RCV_W_LENERR(R6),-      ; Save length and error bits
                   14 A2        20E1  6769                   CXB$W_R_LENERR(R2)      ;
                   F000 8F  AB  20E3  6770          BICW3   #^C<RCV_ERR_M_MLEN>,-   ; Store length in CXB format
                   06 A6        20E7  6771                   RCV_W_LENERR(R6),-      ;  also
                   0C A2        20E9  6772                   CXB$W_LENGTH(R2)        ;
                   05 A6    90  20EB  6773          MOVB    RCV_B_FLAGS(R6),-       ; Save flags byte
                   0B A2        20EE  6774                   CXB$B_R_FLAGS(R2)       ;
                   02B8 C4    95  20F0  6775          TSTB    CDB_B_AQUOTA(R4)        ; Are running on extra QUOTA?
                         06    12  20F4  6776          BNEQ    13$                     ; Br if not
                   02B8 C4    97  20F6  6777          DECB    CDB_B_AQUOTA(R4)        ; Else, decrement extra QUOTA
```

```
                     07   11  20FA  6778          BRB     16$                     ; Continue
                                20FC  6779
                00BB C4   A0  20FC  6780 13$:     ADDW    CDB_W_BSZ(R4),-         ; Replenish CDB quota
                00BA C4       2100  6781                   CDB_W_QUOTA(R4)         ;
            51    22 A2   9A  2103  6782 16$:     MOVZBL  CXB$B_XE_SLOT(R2),R1    ; Get mapping slot number used
                              2107  6783 ;
                              2107  6784 ;       Conditionalized use of buffered data paths
                              2107  6785 ;
                              2107  6786         .IF DF  RCV_DATAP
                              2107  6787         PUSHQ   R2                      ; Save R2, R3
                              2107  6788         MOVL    CDB_L_UCB0(R4),R5       ; Get address of UCB #0
                              2107  6789         MOVL    UCB$L_CRB(R5),R3        ; Get CRB address
                              2107  6790         MOVAL   CDB_L_RCVMAP(R4)[R1],R2 ; Get map info slot address
                              2107  6791         MOVL    (R2),CRB$L_INTD+VEC$W_MAPREG(R3) ; Copy data path number to CRB
                              2107  6792
                              2107  6793         ASSUME  VEC$B_DATAPATH EQ VEC$W_MAPREG+3
                              2107  6794         CLRB    3(R2)                   ; Clear data path number used
                              2107  6795
                              2107  6796         .IFTF   ; DF RCV_DATAP
                              2107  6797
                              2107  6798         CLRBIT  R1,CDB_B_RCVMAP(R4)     ; Clear in use flag
                              210C  6799
                              210C  6800         .IFT    ; DF RCV_DATAP
                              210C  6801
                              210C  6802         PURDPR                          ; Purge the data path (ignore errors)
                              210C  6803         RELDPR                          ; Release the data path
                              210C  6804         POPQ    R2                      ; Restore R2, R3
                              210C  6805
                              210C  6806         .ENDC   ; DF RCV_DATAP
                              210C  6807
                50    01      9A  210C  6808         MOVZBL  S^#SS$_NORMAL,R0        ; Return success!
                              05  210F  6809 20$:     RSB
```

```
                        2110  6811          .SBTTL  MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
                        2110  6812 ;++
                        2110  6813 ; MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
                        2110  6814 ;
                        2110  6815 ; FUNCTIONAL DESCRIPTION:
                        2110  6816 ;
                        2110  6817 ; This routine is called to process a remote request to read the LINE counters.
                        2110  6818 ;
                        2110  6819 ; Inputs:
                        2110  6820 ;
                        2110  6821 ;       R2 = Address of the received message
                        2110  6822 ;       R4 = CDB address
                        2110  6823 ;
                        2110  6824 ; Outputs:
                        2110  6825 ;
                        2110  6826 ;       R0,R1 are destroyed
                        2110  6827 ;       R4 is preserved
                        2110  6828 ;--
                        2110  6829
                        2110  6830 MOP_CTR_REQUEST::                          ; Process a read counters request
          10 A2   D5    2110  6831          TSTL    CXB$L_LINK(R2)            ; Is this a chained message?
             30   12    2113  6832          BNEQ    80$                       ; Br if yes, return packets
                  B1    2115  6833          CMPW    #IRP$C_LENGTH-            ; Is a receive buffer large enough?
                        2116  6834                  +CXB$C_HEADER+3-
                        2116  6835                  +MOP_CTR_SIZE+8,-
    08 A2   0158 8F     2116  6836                  CXB$W_SIZE(R2)
             28   1A    211B  6837          BGTRU   80$                       ; Br if no, ignore message
       55    00C8 C4    211D  6838          MOVL    CDB_L_UCB0(R4),R5         ; Get address of UCB #0
         50     3E A2   7D    2122  6839          MOVQ    CXB$G_R_SRC(R2),R0        ; Save source node address
                47 A2   B0    2126  6840          MOVW    CXB$T_R_USERDAT+1(R2),-; Copy the request ID
                010D C2       2129  6841                  IRP$C_LENGTH+CXB$C_HEADER+1(R2) ;  to the Message block
                   069F 30    212C  6842          BSBW    BLD_IRP_CO                ; Turn the message block into an IRP
           40 A3    50   7D    212F  6843          MOVQ    R0,IRP$Q_STATION(R3)      ; Set the return node address
           14 A3    54   D0    2133  6844          MOVL    R4,IRP$L_ASTPRM(R3)       ; Save the CDB address
                    19   90    2137  6845          MOVB    #XE_FC_V_MOPCTR,-         ; Request the counters
                 21 A3         2139  6846                  IRP$B_XE_FUNC(R3)         ;
           3A A3    01   B0    213B  6847          MOVW    S^#SS$_NORMAL,IRP$W_XE_STATUS(R3) ; Assume success
        0C A3    51'AF   9E    213F  6848          MOVAB   B^MOP_CTR_BUILD,IRP$L_PID(R3) ; Store return address
                         05    2144  6849          RSB                               ; Queue the request to the DEUNA/DELUA
                               2145  6850 ;
                               2145  6851 ; Error - returns packets to receive queue
                               2145  6852 ;
                 10 A2   DD    2145  6853 80$:     PUSHL   CXB$L_LINK(R2)            ; Save next in chain
                  F264   30    2148  6854          BSBW    ADDRCVLIST                ; Add buffer to receive list
                     52 8ED0   214B  6855          POPL    R2                        ; Restore next buffer
                     F5   12    214E  6856          BNEQ    80$                       ; Loop if more
                          05    2150  6857          RSB                               ; Return to caller
                               2151  6858
```

```
                     2151  6860          .SBTTL  MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
                     2151  6861 ;++
                     2151  6862 ; MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
                     2151  6863 ;
                     2151  6864 ; FUNCTIONAL DESCRIPTION:
                     2151  6865 ;
                     2151  6866 ; This routine is called to build the return message in response to a
                     2151  6867 ; MOP read counters request.
                     2151  6868 ;
                     2151  6869 ; Inputs:
                     2151  6870 ;
                     2151  6871 ;       R5 = IRP address
                     2151  6872 ;
                     2151  6873 ; Outputs:
                     2151  6874 ;
                     2151  6875 ;       R4 = CDB address
                     2151  6876 ;       R5 = UCB address of UNIT 0
                     2151  6877 ;
                     2151  6878 ;       R0,R1 are destroyed
                     2151  6879 ;--
                     2151  6880
                     2151  6881 MOP_CTR_BUILD:                          ; Build the MOP counter return msg
        03 38 A5  E8 2151  6882          BLBS    IRP$L_IOST1(R5),10$     ; Br if error
            0619  31 2155  6883          BRW     DELETE_BLOCK            ; Else, deallocate the block
                     2158  6884
          53  55  D0 2158  6885 10$:     MOVL    R5,R3                   ; Copy IRP address
        54  14 A3  D0 215B  6886          MOVL    IRP$L_ASTPRM(R3),R4     ; Get the CDB address
           01F8 8F  BB 215F  6887          PUSHR   #^M<R3,R4,R5,R6,R7,R8>  ; Save registers
          56  54  D0 2163  6888          MOVL    R4,R6                   ; Copy CDB address
    53  010C C3  9E 2166  6889          MOVAB   IRP$C_LENGTH+CXB$C_HEADER(R3),R3 ; Point to start of message block
          83  0B  90 216B  6890          MOVB    #NI_CTR_REPLY,(R3)+     ; Set function to reply
              83  B5 216E  6891          TSTW    (R3)+                   ; Skip request ID (filled in earlier)
    57  E01D CF  9E 2170  6892          MOVAB   MOPCTRTAB,R7            ; Get address of MOP counters
          58  87  9A 2175  6893 20$:     MOVZBL  (R7)+,R8                ; Get the offset to the counter desired
              1E  13 2178  6894          BEQL    30$                     ; Br if end of table
          52  0CEA C6  9E 217A  6895          MOVAB   CDB_G_CUDB(R6),R2       ; Get offset to counter block
          58  52  C0 217F  6896          ADDL    R2,R8                   ; Compute address of the counters
52  67  07  00  EF 2182  6897          EXTZV   #0,#7,(R7),R2           ; Get the width field without bitmap
    63  68  52  28 2187  6898          MOVC3   R2,(R8),(R3)            ; Copy the counter
          51  87  9A 218B  6899          MOVZBL  (R7)+,R1                ; Get the width field again
        E3 51  07  E1 218E  6900          BBC     #7,R1,20$               ; Br if no BITMAP to return
        83  FE A8  B0 2192  6901          MOVW    -2(R8),(R3)+            ; Else, store the BITMAP
              DD  11 2196  6902          BRB     20$                     ; Loop for next counter
                     2198  6903 ;
                     2198  6904 ; Now copy the counters that the table can't handle
                     2198  6905 ;
    83  00D0 C6  B0 2198  6906 30$:     MOVW    CDB_W_UFDCTR(R6),(R3)+  ; Copy the UFD counter
    83  0102 C6  B0 219D  6907          MOVW    CDB_G_CUDB+24(R6),(R3)+ ; Copy the OVR counter
    83  0104 C6  B0 21A2  6908          MOVW    CDB_G_CUDB+26(R6),(R3)+ ; Copy the LBE counter
    83  00D4 C6  B0 21A7  6909          MOVW    CDB_W_UBUCTR(R6),(R3)+  ; Copy the UBU counter
           01F8 8F  BA 21AC  6910          POPR    #^M<R3,R4,R5,R6,R7,R8>  ; Restore registers
  0C A3  2771'CF  9E 21B0  6911          MOVAB   W^DELETE_BLOCK,IRP$L_PID(R3) ; Set the return address
          52  00C4 C3  9E 21B6  6912          MOVAB   IRP$C_LENGTH(R3),R2     ; Get address of CXB
            0A A2  1B  90 21BB  6913          MOVB    #DYN$C_CXB,CXB$B_TYPE(R2) ; Make it look like a CXB
           004C 8F  B0 21BF  6914          MOVW    #MOP_CTR_SIZE+8+3,-     ; Set size of transfer
                1A A2      21C3  6915                  CXB$W_BCNT(R2)          ;
                     21C5  6916          ASSUME  CXB$C_HEADER EQ CXB$T_T_DATA+XE_C_HEADER
```

```
                18 A2    3A   B0   21C5  6917          MOVW    #CXB$T_T_DATA,CXB$W_BOFF(R2) ; Set offset to start of data
                24 A2    53   D0   21C9  6918          MOVL    R3,CXB$L_T_IRP(R2)           ; Save IRP address in CXB
             28 A2    40 A3   7D   21CD  6919          MOVQ    IRP$Q_STATION(R3),CXB$Q_STATION(R2) ; Set STATION in CXB
                20 A2    07   90   21D2  6920          MOVB    #XE_FC_V_XMIT,CXB$B_XE_FUNC(R2) ; Set function request in CXB
                52    3A A2   9E   21D6  6921          MOVAB   CXB$T_T_DATA(R2),R2          ; Set R2 to start of data
                   40 A3   7D   21DA  6922          MOVQ    IRP$Q_STATION(R3),-          ; Set th destination address
                      62        21DD  6923                  XBUF_G_DEST(R2)
                   0260 8F   B0   21DE  6924          MOVW    #NI_CTR_PROTYP,-             ; Store the protocol type
                     0C A2        21E2  6925                  XBUF_W_TYPE(R2)
                55    1C A3   D0   21E4  6926          MOVL    IRP$C_OCB(R3),R5             ; Get back the UCB address
                08 A2    55   D0   21E8  6927          MOVL    R5,XBUF_L_UCB(R2)            ; Save UCB address
          0080 D4    00C4 C3   0E   21EC  6928          INSQUE  IRP$C_LENGTH(R3),@CDB_Q_XMTREQ+4(R4) ; Insert request on request que
                   E2B2   31   21F3  6929          BRW     XMT_ACT_START                ; Startup the reply
                          21F6  6930
```

```
                                21F6  6932          .SBTTL  REG_DUMP - DEUNA/DELUA ERROR LOG AND DIAGNOSTICS REGISTER DUMP
                                21F6  6933 ;++
                                21F6  6934 ; REG_DUMP - DIAGNOSTICS REGISTER DUMP ROUTINE
                                21F6  6935 ;
                                21F6  6936 ; FUNCTIONAL DESCRIPTION:
                                21F6  6937 ;
                                21F6  6938 ; This routine is used to return the DEUNA/DELUA error log and diagnostics
                                21F6  6939 ; buffer on error or diagnostic QIO function.
                                21F6  6940 ;
                                21F6  6941 ; Inputs:
                                21F6  6942 ;
                                21F6  6943 ;       R0 = Address of the buffer
                                21F6  6944 ;       R4 = CDB address
                                21F6  6945 ;       R5 = UCB address of the unit
                                21F6  6946 ;
                                21F6  6947 ; Outputs:
                                21F6  6948 ;
                                21F6  6949 ;       R0,R1 are destroyed
                                21F6  6950 ;       R4,R5 are preserved
                                21F6  6951 ;--
                                21F6  6952 REG_DUMP::
                                21F6  6953          ASSUME  DIAG_L_EXTRA EQ 32
                                21F6  6954          ASSUME  DIAG_Q_PCSR EQ DIAG_L_EXTRA+4
                                21F6  6955          ASSUME  DIAG_Q_PCBB EQ DIAG_Q_PCSR+8
                                21F6  6956          ASSUME  DIAG_W_ERR EQ DIAG_Q_PCBB+8
                                21F6  6957          ASSUME  DIAG_G_HWA EQ DIAG_W_ERR+2
                                21F6  6958          ASSUME  DIAG_W_ERR2 EQ DIAG_G_HWA+6
          05   E1               21F6  6959          BBC     #CDB_STS_V_RCBI,-       ; Br if no RCBI error
06 025A C4                     21F8  6960                  CDB_B_STS(R4),10$       ;
  0400 8F    A8                21FC  6961          BISW    #XE_PCSR0_M_RCBI,-      ; Else, indicate the error
    10 A4                      2200  6962                  CDB_L_PCSR0(R4)         ;
  80   0A    9A                2202  6963 10$:     MOVZBL  #DIAG_C_EXTRA,(R0)+     ; Insert number of returned long words
                               2205  6964          ;
                               2205  6965          ; If we are in the RUN state, then clear bit 13 which toggles
                               2205  6966          ; every second. This is to give the user a consistent "view" of
                               2205  6967          ; the UNA state!
                               2205  6968          ;
          10   EF               2205  6969          EXTZV   #XE_PCSR1_V_STATE+16,-  ; Get the current state
          04                   2207                        #XE_PCSR1_S_STATE,-     ;   (from high word)
    51  10 A4                   2208  6970                  CDB_L_PCSR0(R4),R1      ;     in R1
      51   03   91               220B  6971          CMPB    #XE_STAT_V_RUN,R1       ; Are we in the RUN state?
           05   12               220E  6973          BNEQ    20$                     ; Br if no, don't clear BIT13
                               2210  6974          CLRBIT  #13+16,CDB_L_PCSR0(R4)  ; Else, clear BIT13 (in high word!)
  80  10 A4    D0               2215  6975 20$:     MOVL    CDB_L_PCSR0(R4),(R0)+   ; Insert last CSR contents
           80   D4               2219  6976          CLRL    (R0)+                   ; Nothing from last port CSRs
80   00D6 C4    7D               221B  6977          MOVQ    CDB_G_PCBB(R4),(R0)+    ; Copy last PCBB info
80   00CD C4    9B               2220  6978          MOVZBW  CDB_B_DIAG1(R4),(R0)+   ; Return error flags info
80   0269 C4    D0               2225  6979          MOVL    CDB_G_HWA(R4),(R0)+     ; Return hardware physical address
80   026D C4    B0               222A  6980          MOVW    CDB_G_HWA+4(R4),(R0)+   ;                 ...
80   00CE C4    B0               222F  6981          MOVW    CDB_W_DIAG2(R4),(R0)+   ; Return extra error info
                   05   2234  6982          RSB
```

```
                     2235  6984          .SBTTL  RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
                     2235  6985 ;++
                     2235  6986 ; RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
                     2235  6987 ;
                     2235  6988 ; Functional description:
                     2235  6989 ;
                     2235  6990 ; This routine is entered when the RESTART delta time has expired. The action
                     2235  6991 ; is to check if the specified unit has been restarted, if so, then it is
                     2235  6992 ; automatically restarted.
                     2235  6993 ;
                     2235  6994 ; Inputs:
                     2235  6995 ;
                     2235  6996 ;       R4 = CDB address
                     2235  6997 ;       R5 = TQE address (but must be at least as long as an IRP)
                     2235  6998 ;
                     2235  6999 ;       IPL = IPL$_TIMER
                     2235  7000 ;
                     2235  7001 ; Implicit inputs:
                     2235  7002 ;
                     2235  7003 ;       IRP$L_RBOFF(R5) = UCB address for UNIT
                     2235  7004 ;
                     2235  7005 ; Outputs:
                     2235  7006 ;
                     2235  7007 ;       R0-R3 are destroyed.
                     2235  7008 ;--
                     2235  7009          .ENABL  LSB
                     2235  7010 RESTART_ROUT::                            ; Process expiration of restart timer
          55   DD    2235  7011          PUSHL   R5                      ; Save R5
                     2237  7012          ASSUME  IRP$L_RBOFF GT TQE$C_LENGTH
    52    30 A5   9E 2237  7013          MOVAB   TQE$C_LENGTH(R5),R2     ; Point to IRP portion of TQE
55     0098 C2   D0 223B  7014          MOVL    IRP$L_RBOFF(R2),R5      ; Reset R5 to UCB address
                     2240  7015 ;
                     2240  7016 ; Turn TQE into an IRP
                     2240  7017 ;
          53   82   7E 2240  7018          MOVAQ   (R2)+,R3                ; Copy IRP address, skip to size field
                     2243  7019          ASSUME  IRP$W_SIZE EQ 8
                     2243  7020          ASSUME  IRP$B_TYPE EQ IRP$W_SIZE+2
                     2243  7021          ASSUME  IRP$B_RMOD EQ IRP$B_TYPE+1
          52   02   C0 2243  7022          ADDL    #2,R2                   ; Skip size field
          82   0A   B0 2246  7023          MOVW    #DYN$C_IRP,(R2)+        ; Set type to IRP
                     2249  7024          ASSUME  IRP$L_PID EQ IRP$B_RMOD+1
    82   228E'CF   9E 2249  7025          MOVAB   W^RETURN_IRP,(R2)+      ; Set return address form IOPOST
                     224E  7026          ASSUME  IRP$L_AST EQ IRP$L_PID+4
                     224E  7027          ASSUME  IRP$L_ASTPRM EQ IRP$L_AST+4
               82    7C 224E  7028          CLRQ    (R2)+                   ; Clear AST, ASTPRM
                     2250  7029          ASSUME  IRP$L_WIND EQ IRP$L_ASTPRM+4
                     2250  7030          ASSUME  IRP$L_UCB EQ IRP$L_WIND+4
               82    D4 2250  7031          CLRL    (R2)+                   ; Clear WIND
          82   55   D0 2252  7032          MOVL    R5,(R2)+                ; Set UCB address
                     2255  7033          ASSUME  IRP$W_FUNC EQ IRP$L_UCB+4
                     2255  7034          ASSUME  IRP$B_XE_FUNC EQ IRP$W_FUNC+1
                     2255  7035          ASSUME  IRP$B_EFN EQ IRP$W_FUNC+2
                     2255  7036          ASSUME  IRP$B_PRI EQ IRP$B_EFN+1
                     2255  7037          ASSUME  IRP$L_IOSB EQ IRP$B_PRI+1
    82   1700 8F   B0 2255  7038          MOVW    #<XE_FC_V_RESTART@8>,(R2)+ ; Set function request
               82    B4 225A  7039          CLRW    (R2)+                   ; Clear EFN, PRI
               82    D4 225C  7040          CLRL    (R2)+                   ; Clear IOSB
```

```
                     225E  7041          ASSUME  IRP$W_CHAN EQ IRP$L_IOSB+4
                     225E  7042          ASSUME  IRP$W_STS EQ IRP$W_CHAN+2
                     225E  7043          ASSUME  IRP$L_SVAPTE EQ IRP$W_STS+2
           82   7C   225E  7044          CLRQ    (R2)+                   ; Clear CHAN, STS, SVAPTE
                     2260  7045          ASSUME  IRP$W_BOFF EQ IRP$L_SVAPTE+4
                     2260  7046          ASSUME  IRP$W_BCNT EQ IRP$W_BOFF+2
                     2260  7047          ASSUME  IRP$W_BCNT EQ IRP$L_BCNT
                     2260  7048          ASSUME  IRP$L_MEDIA EQ IRP$Q_BCNT+6
           82   7C   2260  7049          CLRQ    (R2)+                   ; Clear BOFF, BCNT
        82 00   3C   2262  7050          MOVZWL  #<XE_FC_V_INIT@8>,(R2)+ ; Set MEDIA
           82   D4   2265  7051          CLRL    (R2)+                   ; Clear MEDIA+4
                     2267  7052 ;
                     2267  7053 ; RESTART the UNIT
                     2267  7054 ;
                     2267  7055          DSBINT  UCB$B_FIPL(R5)          ; Raise IPL
      4000 8F   AA   226E  7056          BICW    #UCB$M_XE_INTERLOCK,-   ; Clear the RESTART interlock
        68 A5        2272  7057                  UCB$W_DEVSTS(R5)        ;
           00   E2   2274  7058          BBSS    #UCB$V_XE_INITED,-      ; Br if unit already inited
     09 68 A5        2276  7059                  UCB$W_DEVSTS(R5),10$    ;
         ECC4   30   2279  7060          BSBW    START                   ; Start protocol
        08 50   E8   227C  7061          BLBS    R0,30$                  ; Br if success
                     227F  7062
         ECB6   30   227F  7063          BSBW    STOP                    ; Shutdown unit
     55    53   D0   2282  7064 10$:     MOVL    R3,R5                   ; Point R5 to IRP
           07   10   2285  7065          BSBB    RETURN_IRP              ; Return the IRP
                     2287  7066 30$:     ENBINT                          ; Re-enable interrupts
           55 8ED0  228A  7067          POPL    R5                      ; Restore R5
                05   228D  7068          RSB                             ; Return to caller
                     228E  7069
                     228E  7070 RETURN_IRP:
     50    D0 A5   9E   228E  7071          MOVAB   -IQE$C_LENGTH(R5),R0    ; Get address of start of structure
00000000'GF   17   2292  7072          JMP     G^COM$DRVDEALMEM        ; Deallocate the IRP
```

```
                     2298  7074          .SBTTL  TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
                     2298  7075 ;++
                     2298  7076 ; TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
                     2298  7077 ;
                     2298  7078 ; Functional description:
                     2298  7079 ;
                     2298  7080 ; This routine is entered when the TQE delta time has expired. The action is to
                     2298  7081 ; check all timer cells and shut down the controller if any have expired.
                     2298  7082 ;
                     2298  7083 ; Inputs:
                     2298  7084 ;
                     2298  7085 ;       R4 = CDB address
                     2298  7086 ;       R5 = TQE address
                     2298  7087 ;
                     2298  7088 ;       IPL = IPL$_TIMER
                     2298  7089 ;
                     2298  7090 ; Outputs:
                     2298  7091 ;
                     2298  7092 ;       R0-R3 are destroyed.
                     2298  7093 ;       R4,R5 are preserved
                     2298  7094 ;--
                     2298  7095          .ENABL  LSB
                     2298  7096 TQE_TIMER::                                ; Process expiration of TQE timer
 50   00C8 C4   D0   2298  7097          MOVL    CDB_L_UCBO(R4),R0          ; Get first UCB address
                     229D  7098          DSBINT  UCB$B_DIPL(R0)             ; Sync access to UCB and CDB
      00   E1   22A4  7099          BBC     #CDB_STS_V_INITED,-        ; Br if NOT inited
 71 025A C4        22A6  7100                  CDB_B_STS(R4),STOP_TQE
      0222 C4   95  22AA  7101          TSTB    CDB_B_TIM_XMT(R4)          ; Is the xmit timer going?
           06   13  22AE  7102          BEQL    10$                        ; Br if not
      0222 C4   97  22B0  7103          DECB    CDB_B_TIM_XMT(R4)          ; Timer expired?
           5A   13  22B4  7104          BEQL    TIMOUT                     ; Br if yes
      0223 C4   95  22B6  7105 10$:     TSTB    CDB_B_TIM_DNI(R4)          ; Is the DNI timer going?
           06   13  22BA  7106          BEQL    20$                        ; Br if not
      0223 C4   97  22BC  7107          DECB    CDB_B_TIM_DNI(R4)          ; Timer expired?
           4E   13  22C0  7108          BEQL    TIMOUT                     ; Br if yes
           0C   E1  22C2  7109 20$:     BBC     #XM$V_STS_BUFFAIL,-        ; Br if NO buffer allocation failures
 07 00C4 C4        22C4  7110                  CDB_L_DEVDEPEND(R4),30$ ;
                   22C8  7111          SETIPL  CDB_B_FIPL(R4)             ; Else, sync access to UCB & CDB
      FODE   30    22CC  7112          BSBW    FILCRCVLIST                ; And try to replenish receive buffers
                   22CF  7113 30$:     ENBINT                             ; Restore IPL
                   22D2  7114
                   22D2  7115          .IF DF  POINT
                   22D2  7116          PUSHQ   R4                         ;% Save R4, R5
 55   00C8 C4   D0   22D5  7117          MOVL    CDB_L_UCBO(R4),R5          ;% Get UCB address of unit 0
                   22DA  7118          DSBINT  UCB$B_FIPL(R5)             ;% Sync access to UCB
  55    30 A5   D0   22E1  7119 50$:     MOVL    UCB$L_LINK(R5),R5          ;% Travel down UCBs
           22   13  22E5  7120          BEQL    70$                        ;% Br if end of list
                   22E7  7121          ASSUME  UCB$V_XE_INITED EQ 0
    F6 68 A5   E9  22E7  7122          BLBC    UCB$W_DEVSTS(R5),50$       ;% Br if not inited
           04   E1  22EB  7123          BBC     #UCB$V_XE_RUN,-            ;% Br if not running yet
    F1 68 A5       22ED  7124                  UCB$W_DEVSTS(R5),50$       ;%
           00   91  22F0  7125          CMPB    #NMA$C_LINPR_POI,-         ;% Are we in point to point mode?
     00D8 C5       22F2  7126                  UCB$B_XE_PRO(R5)           ;%
           EA   12  22F5  7127          BNEQ    50$                        ;% Br if not
                B3  22F7  7128          BITW    #UCB$M_XE_START!-          ;% Br if not in startup or
                   22F8  7129                   UCB$M_XE_STACK,-           ;%  stack wait states
68 A5    0060 8F    22F8  7130                  UCB$W_DEVSTS(R5)           ;%
```

```
                E2   13   22FD  7131          BEQL    50$                             ;%
52    017C      C5   D0   22FF  7132          MOVL    UCB$L_XE_STIRP(R5),R2           ;% Get startup IRP addresss
         0473   30   2304  7133          BSBW    BLD_STRT_IRP                    ;% Build the startup IRP
                D8   11   2307  7134          BRB     50$                             ;% Look for more
                          2309  7135  70$:    ENBINT                                  ;% Re-enable interrupts
                          230C  7136          POPQ    R4                              ;% Restore R4, R5
                          230F  7137          .ENDC
                          230F  7138
                05        230F  7139          RSB                                     ; Return to caller
                          2310  7140
                          2310  7141  TIMOUT: PUSHQ   R4                              ; Save R4, R5
          55    50   D0   2313  7142          MOVL    R0,R5                           ; Copy address of UCB0
                31   10   2316  7143          BSBB    DEV_TIMEOUT                     ; Else, timeout has occurred
                          2318  7144          POPQ    R4                              ; Restore R4, R5
                          231B  7145
                          231B  7146  STOP_TQE:                                       ; Stop the TQE
       0B A5      04   8A   231B  7147          BICB    #TQE$M_REPEAT,TQE$B_RQTYPE(R5) ; Stop the timer
                  10   8A   231F  7148          BICB    #CDB_STS_M_TIMER,-              ; Indicate that timer is stopped
          025A    C4        2321  7149                  CDB_B_STS(R4)                   ;
                  04   8A   2324  7150          BICB    #DPT$M_NOUNLOAD,-               ; OKAY to unload the driver now
00000000D'        EF        2326  7151                  DPT$TAB+DPT$B_FLAGS             ;
                  A2   11   232B  7152          BRB     30$                             ; Leave
                          232D  7153          .DSABL  LSB
```

```
                        232D  7155          .SBTTL  TIMEOUT - TIMEOUT SERVICE ROUTINE
                        232D  7156 ;++
                        232D  7157 ; TIMEOUT - TIMEOUT SERVICE ROUTINE
                        232D  7158 ;
                        232D  7159 ; Functional description:
                        232D  7160 ;
                        232D  7161 ; This routine is entered on device timeout. The action is to
                        232D  7162 ; shut the unit down.
                        232D  7163 ;
                        232D  7164 ; Inputs:
                        232D  7165 ;
                        232D  7166 ;       R5 = UCB ADDRESS
                        232D  7167 ;
                        232D  7168 ;       IPL = DIPL
                        232D  7169 ;
                        232D  7170 ; Outputs:
                        232D  7171 ;
                        232D  7172 ;       R3,R4 are destroyed.
                        232D  7173 ;       R5 is preserved
                        232D  7174 ;--
                        232D  7175          .ENABL  LSB
                        232D  7176 TIMEOUT::                                    ; Timeout or powerfail
    64 A5    03    AA   232D  7177          BICW    #UCB$M_TIM!UCB$M_INT,UCB$W_STS(R5) ; Disable timer
                        2331  7178
                        2331  7179          ASSUME  UCB$V_XE_INITED  EQ  0
    38 68 A5    E9      2331  7180          BLBC    UCB$W_DEVSTS(R5),20$        ; Br if not inited
          05    E1      2335  7181          BBC     #UCB$V_POWER,-              ; Branch if device timeout
       64 A5            2337  7182                  UCB$W_STS(R5),-
          0F            2339  7183                  DEV_TIMEOUT
54  0180 C5    D0      233A  7184          MOVL    UCB$L_XE_CDB(R5),R4         ; Get CDB address
          2C    13      233F  7185          BEQL    20$                         ; Just return if no CDB
    80 8F    88         2341  7186          BISB2   #CDB_STS_M_POWER,-          ; Set the CDB power failure bit
  025A C4               2344  7187                  CDB_B_STS(R4)
          17    11      2347  7188          BRB     10$                         ; Go handle the error
                        2349  7189
                        2349  7190 DEV_TIMEOUT::                               ; Hardware device timeout
    54  24 A5    D0      2349  7191          MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
                        234D  7192          ASSUME  IDB$L_CSR EQ 0
    53  2C B4    D0      234D  7193          MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R3 ; Get CSR address
    63    20     B0      2351  7194          MOVW    #XE_PCSR0_M_RSET,PCSR0(R3)  ; Stop the device
    54  10 A4    D0      2354  7195          MOVL    CRB$L_AUXSTRUC(R4),R4       ; Get CDB address
          13    13      2358  7196          BEQL    20$                         ; Br if no CDB
                        235A  7197          SETBIT  #XM$V_STS_TIMO,CDB_L_DEVDEPEND(R4) ; Set error status
53  0100 8F    B0      2360  7198 10$:      MOVW    #XE_PCSR0_M_USCI,R3         ; Indicate fatal error
          55    DD      2365  7199          PUSHL   R5                          ; Save UCB address
        F59A    30      2367  7200          BSBW    SCHED_FORK                  ; Schedule a fork process
          55 8ED0       236A  7201          POPL    R5                          ; Restore UCB address
                05      236D  7202 20$:      RSB                                 ; Return to caller
                        236E  7203          .DSABL  LSB
```

```
                                236E  7205          .SBTTL  ALLOC_CDB - ALLOCATE THE CDB
                                236E  7206 ;++
                                236E  7207 ; ALLOC_CDB - ALLOCATE THE CDB
                                236E  7208 ;
                                236E  7209 ; Functional description:
                                236E  7210 ;
                                236E  7211 ; This routine allocates and initializes the CDB.
                                236E  7212 ;
                                236E  7213 ; Inputs:
                                236E  7214 ;
                                236E  7215 ;       R5 = UCB address
                                236E  7216 ;
                                236E  7217 ; Outputs:
                                236E  7218 ;
                                236E  7219 ;       R0 = Status return for request
                                236E  7220 ;
                                236E  7221 ;       All other registers are preserved.
                                236E  7222 ;
                                236E  7223 ;--
                                236E  7224
                                236E  7225 ALLOC_CDB:                                   ; Allocate a CDB
          3E   BB               236E  7226          PUSHR   #^M<R1,R2,R3,R4,R5>         ; Save registers
  51   02BA 8F   3C             2370  7227          MOVZWL  #CDB_C_LENGTH,R1            ; Get size of CDB allocation
   00000000'GF   16             2375  7228          JSB     G^EXE$ALONONPAGED           ; Try to allocate CDB
          47 50   E9            2378  7229          BLBC    R0,90$                      ; Br if error
                                237E  7230
    54    24 A5   D0            237E  7231          MOVL    UCB$L_CRB(R5),R4            ; Get CRB address
    10 A4    52   D0            2382  7232          MOVL    R2,CRB$L_AUXSTRUC(R4)       ; Store CDB address
    54    28 A5   D0            2386  7233          MOVL    UCB$L_DDB(R5),R4            ; Get DDB address
    54    04 A4   D0            238A  7234          MOVL    DDB$L_UCB(R4),R4            ; Get UCB0 address
       53    54   D0            238E  7235          MOVL    R4,R3                       ; Save UCB0 address
  0180 C4    52   D0            2391  7236 10$:     MOVL    R2,UCB$L_XE_CDB(R4)         ; Store CDB address
          0269 C2   9E          2396  7237          MOVAB   CDB_G_HWA(R2),-             ; Store address of NI device's
          0090 C4               239A  7238                  UCB$L_NI_HWAPTR(R4)         ; unique hardware address
    54    30 A4   D0            239D  7239          MOVL    UCB$L_LINK(R4),R4           ; Position to next UCB and continue
             EE   12            23A1  7240          BNEQ    10$                         ; if there is a next UCB
                                23A3  7241
                                23A3  7242 ;
                                23A3  7243 ; Initialize CDB
                                23A3  7244 ;
                                23A3  7245 ;   The PADDING MODE and the ECHO MODE of the DEUNA/DELUA will default to
                                23A3  7246 ;   the enabled (ON) state.
                                23A3  7247 ;
                                23A3  7248          ASSUME  NMA$C_STATE_ON EQ 0
                                23A3  7249
                                23A3  7250          PUSHQ   R2                          ; Save CDB, UCB0 address
62   51   00   62   00   2C    23A6  7251          MOVC5   #0,(R2),#0,R1,(R2)          ; Zero the structure
                                23AC  7252          POPQ    R2                          ; Restore CDB, UCB0 address
                                23AF  7253
          00C8 C2   53   D0     23AF  7254          MOVL    R3,CDB_L_UCB0(R2)           ; Setup address of UCB unit 0
                                23B4  7255 ;
                                23B4  7256 ;  Initialize fork block in CDB
                                23B4  7257 ;
                                23B4  7258          ASSUME  CDB_L_FQFL EQ 0
                                23B4  7259          ASSUME  CDB_L_FQBL EQ CDB_L_FQFL+4
                   82   7C      23B4  7260          CLRQ    (R2)+                       ; Skip link pointers
                                23B6  7261          ASSUME  CDB_W_SIZE EQ CDB_L_FQBL+4
```

```
                                23B6  7262          ASSUME  CDB_B_TYPE EQ CDB_W_SIZE+2
                                23B6  7263          ASSUME  CDB_B_FIPL EQ CDB_B_TYPE+1
        82   083302BA 8F    D0  23B6  7264          MOVL    #<<^IPL$_XE_FIPL@8>!DYN$C_CDB>@16>!-; Set size, type and FIPL
                                23BD  7265                  CDB_C_LENGTH,(R2)+
                                23BD  7266          ASSUME  CDB_L_FPC EQ CDB_B_FIPL+1
            82   F560 CF    9E  23BD  7267          MOVAB   FORK_PROC,(R2)+         ; Set fork process address
                 50   01    9A  23C2  7268          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
                      3E    BA  23C5  7269 90$:     POPR    #^M<R1,R2,R3,R4,R5>     ; Restore registers
                            05  23C7  7270          RSB                             ; Return to caller
```

```
                        23C8  7272          .SBTTL  SHUTDOWN_UNA - SHUTDOWN DEUNA/DELUA AND ALL UNITS
                        23C8  7273 ;++
                        23C8  7274 ; SHUTDOWN_UNA - SHUTDOWN DEUNA/DELUA AND ALL UNITS
                        23C8  7275 ;
                        23C8  7276 ; Inputs:
                        23C8  7277 ;
                        23C8  7278 ;       R4 = CDB address
                        23C8  7279 ;       R5 = UCB address of unit #0
                        23C8  7280 ;
                        23C8  7281 ;
                        23C8  7282 ;       IPL = FIPL
                        23C8  7283 ;
                        23C8  7284 ; Outputs:
                        23C8  7285 ;
                        23C8  7286 ;       R3-R5 are preserved.
                        23C8  7287 ;
                        23C8  7288 ;       R0-R2 are destroyed.
                        23C8  7289 ;--
                        23C8  7290
                        23C8  7291 SHUTDOWN_UNA::                          ; Shutdown DEUNA/DELUA
          00    E0      23C8  7292          BBS     #CDB_STS_V_INITED,-    ; Br if DEUNA/DELUA inited
01 025A C4              23CA  7293                  CDB_B_STS(R4),10$      ;
                  05    23CE  7294          RSB                            ; Else, return
                        23CF  7295
     00E8 8F    BB      23CF  7296 10$:     PUSHR   #^M<R3,R5,R6,R7>       ; Save registers
                        23D3  7297 ;
                        23D3  7298 ; Shutdown DEUNA/DELUA and reset controller status
                        23D3  7299 ;
     52   24 A5    D0   23D3  7300          MOVL    UCB$L_CRB(R5),R2       ; Get CRB address
                        23D7  7301          ASSUME  IDB$L_CSR EQ 0
     52   2C B2    D0   23D7  7302          MOVL    @CRB$L_INTD+VEC$L_IDB(R2),R2 ; Get CSR address
                        23DB  7303          DSBINT  UCB$B_DIPL(R5)         ; Raise IPL for master clear
            20    A8    23E2  7304          BISW    #XE_PCSR0_M_RSET,-     ; Disable device
                  62    23E4  7305                  PCSR0(R2)
025A C4     63 8F    8A  23E5  7306          BICB    #^C<CDB_STS_M_FORK_PEND!- ; Reset all but needed bits
                        23EB  7307                  CDB_STS_M_TIMER!CDB_STS_M_POWER!-
                        23EB  7308                  CDB_STS_M_FUNC>,CDB_B_STS(R4) ;
                        23EB  7309          ENBINT                         ; Return to fork level
                        23EE  7310 ;
                        23EE  7311 ; Release the receive and transmit buffer map registers
                        23EE  7312 ;
            57    D4    23EE  7313          CLRL    R7                     ; Init slot number
                        23F0  7314          ASSUME  CDB_L_RCVMAP+<4*MAX_C_RCV> EQ CDB_L_XMTMAP
     56   1C A4    9E   23F0  7315          MOVAB   CDB_L_RCVMAP(R4),R6    ; Get address of mappig slots
     53   24 A5    D0   23F4  7316 20$:     MOVL    UCB$L_CRB(R5),R3       ; Get CRB address
                        23F8  7317          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                        23F8  7318          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
34 A3       86    D0   23F8  7319          MOVL    (R6)+,CRB$L_INTD+VEC$W_MAPREG(R3) ; Set mapping information
            1B    19   23FC  7320          BLSS    30$                    ; Br if none allocated
         37 A3    95   23FE  7321          TSTB    CRB$L_INTD+VEC$B_DATAPATH(R3) ; Is there a datapath?
            0C    13   2401  7322          BEQL    25$                    ; Br if not - don't do purge or release
                       2403  7323          PURDPR                         ; Purge the data path
                       2409  7324          RELDPR                         ; Release the data path
                       240F  7325 25$:     RELMPR                         ; Release the map register
FC A6       01    CE   2415  7326          MNEGL   #1,-4(R6)              ; Reset mapping info
                       2419  7327 30$:     CLRBIT  R7,CDB_B_RCVMAP(R4)    ; Clear mapping slot flag
D2 57       0C    F2   241E  7328          AOBLSS  #MAX_C_RCV+MAX_C_XMT,R7,20$ ; Loop if more map registers
```

```
                     2422  7329 ;
                     2422  7330 ; Release the PCBB map registers
                     2422  7331 ;
53   24 A5   D0      2422  7332          MOVL    UCB$L_CRB(R5),R3        ; Get CRB address
                     2426  7333          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                     2426  7334          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
   00BC C4   D0      2426  7335          MOVL    CDB_L_PCBBMAP(R4),-     ; Setup map info in CRB
     34 A3           242A  7336                   CRB$L_INTD+VEC$W_MAPREG(R3) ;
        0A   13      242C  7337          BEQL    50$                     ; Br if none
   00BC C4   D4      242E  7338          CLRL    CDB_L_PCBBMAP(R4)       ; No more mapping info
                     2432  7339          RELMPR                          ; Release the map register
                     2438  7340 50$:     ;
                     2438  7341          ; Deallocate all receive buffers and complete all I/O request packets
                     2438  7342          ;
     80 8F   8A      2438  7343          BICB    #CDB_STS_M_POWER,-      ; Clear the powerfailure bit (if it had
   025A C4           243B  7344                  CDB_B_STS(R4)           ; been set) at this time
56   7C A4   9E      243E  7345          MOVAB   CDB_Q_QUEUES(R4),R6     ; Get address of first queue listhead
   57    07   3C      2442  7346          MOVZWL  #CDB_C_QUEUES,R7        ; Get number of queues
53   00 B6   0F      2445  7347 60$:     REMQUE  @(R6),R3                ; Get next IRP/BUFFER
        61   1D      2449  7348          BVS     150$                    ; Br if none
0A   0A A3   91      244B  7349          CMPB    IRP$B_TYPE(R3),S^#DYN$C_IRP ; Is this an IRP?
        0A   13      244F  7350          BEQL    70$                     ; Br if yes
1B   0A A3   91      2451  7351          CMPB    IRP$B_TYPE(R3),S^#DYN$C_CXB ; Is this a CXB?
        08   13      2455  7352          BEQL    80$                     ; Br if yes
                     2457  7353          BUG_CHECK NOBUFPCKT,FATAL       ; Else, fatal error
                     245B  7354 ;
                     245B  7355 ; IRP
                     245B  7356 ;
        6C   10      245B  7357 70$:     BSBB    IRP_ABORTED             ; Abort the IRP
        E6   11      245D  7358          BRB     60$                     ; Try for more
                     245F  7359 ;
                     245F  7360 ; CXB
                     245F  7361 ;
                     245F  7362 80$:     $DISPATCH       CXB$B_XE_FUNC(R3),TYPE=B,-
                     245F  7363          <-      ;function       action
                     245F  7364
                     245F  7365                  <XE_FC_V_XMIT   100$>,- ; XMIT request
                     245F  7366                  <XE_FC_V_RECV   140$>,- ; RECV request
                     245F  7367          >
                     2468  7368
                     2468  7369          BUG_CHECK NOBUFPCKT,FATAL       ; Fatal error - not a valid IRP
                     246C  7370
                     246C  7371 ;
                     246C  7372 ; CXB - XMIT request
                     246C  7373 ;
50   24 A3   D0      246C  7374 100$:    MOVL    CXB$L_T_IRP(R3),R0      ; Get IRP address
        07   13      2470  7375          BEQL    110$                    ; Br if none, return buffer
   53   50   D0      2472  7376          MOVL    R0,R3                   ; Copy IRP address
        52   10      2475  7377          BSBB    IRP_ABORTED             ; Abort the IRP
        CC   11      2477  7378          BRB     60$                     ; Try for more
                     2479  7379          ;
                     2479  7380          ;  FFI CXB transmit to complete
                     2479  7381          ;
51   18 A3   3C      2479  7382 110$:    MOVZWL  CXB$W_BOFF(R3),R1       ; Get offset to start of data
   51   53   C0      247D  7383          ADDL    R3,R1                   ; Compute start of Ethernet header
51   0C A1   3C      2480  7384          MOVZWL  XBUF_W_TYPE(R1),R1      ; Get protocol type
     0C78   30      2484  7385          BSBW    MATCH_PROTYP            ; Find the protocol type
```

```
                17 50    E9  2487  7386          BLBC    R0,140$                 ; Br if no UCB, drop buffer
                   54    DD  248A  7387          PUSHL   R4                      ; Save CDB address
          54    0178 C5    D0  248C  7388          MOVL    UCB$L_XE_FFI(R5),R4     ; Get FFI block address
                   0B    13  2491  7389          BEQL    120$                    ; Br if not there, drop buffer
               50   2C    3C  2493  7390          MOVZWL  #SS$_ABORT,R0           ; Else, set return status
                 14 B4    16  2496  7391          JSB     @FFI$L_XMIT_DONE(R4)    ; Complete the XMIT in error
                   54 8ED0  2499  7392          POPL    R4                      ; Restore CDB address
                   A7    11  249C  7393          BRB     60$                     ; Try for more
                         249E  7394 120$:        ;
                         249E  7395              ;  No more UCB for CXB transmit request
                         249E  7396              ;
                   54 8ED0  249E  7397          POPL    R4                      ; Restore CDB address
                         24A1  7398
                         24A1  7399 ;
                         24A1  7400 ; CXB - RECV request
                         24A1  7401 ;
                         24A1  7402
               50   53    D0  24A1  7403 140$:    MOVL    R3,R0                   ; Copy buffer address
          00000000'GF    16  24A4  7404          JSB     G^COM$DRVDEALMEM        ; Deallocate the buffer
                   99    11  24AA  7405          BRB     60$                     ; Try for more
                         24AC  7406              ;
                         24AC  7407              ; Loop to next queue
                         24AC  7408              ;
               56   08    C0  24AC  7409 150$:    ADDL    #8,R6                   ; Skip to next queue listhead
                 93 57    F5  24AF  7410          SOBGTR  R7,60$                  ; Loop if more queues
                         24B2  7411              ;
                         24B2  7412              ; Cleanup all I/O on all UNITS
                         24B2  7413              ;
          55    00C8 C4    D0  24B2  7414          MOVL    CDB_L_UCB0(R4),R5       ; Get UNIT #0 UCB address
                   0B    13  24B7  7415          BEQL    190$                    ; Br if not set yet
            55   30 A5    D0  24B9  7416 170$:    MOVL    UCB$L_LINK(R5),R5       ; Get next unit's address
                   05    13  24BD  7417          BEQL    190$                    ; Br if none
                 0097    30  24BF  7418          BSBW    SHUTDOWN                ; Shutdown the UNIT
                   F5    11  24C2  7419          BRB     170$                    ; Check if more UNIT's
              00E8 8F    BA  24C4  7420 190$:    POPR    #^M<R3,R5,R6,R7>        ; Restore registers
                         05  24C8  7421          RSB                             ; Return to caller
```

```
                        24C9  7423          .SBTTL  IRP_ABORTED - Abort an IRP on one of the CDB queues
                        24C9  7424 ;++
                        24C9  7425 ; IRP_ABORTED - Abort CDB IRP
                        24C9  7426 ;
                        24C9  7427 ; Functional description:
                        24C9  7428 ;
                        24C9  7429 ; The I/O is aborted according to the type of operation requested.  For
                        24C9  7430 ; initialization requests, the SS$_CTRLERR is returned except when the
                        24C9  7431 ; function was a self-test with the TRANSCEIVER DISCONNECTED, then the
                        24C9  7432 ; status is SS$_MEDOFL.  For all other I/O requests, the error returned
                        24C9  7433 ; is SS$_ABORT.
                        24C9  7434 ;
                        24C9  7435 ; Inputs:
                        24C9  7436 ;       R3 = IRP address
                        24C9  7437 ;       R4 = CDB address
                        24C9  7438 ;
                        24C9  7439 ; Outputs:
                        24C9  7440 ;       R4 is preserved
                        24C9  7441 ;       R0,R1,R2,R3,R5 are destroyed.
                        24C9  7442 ;-
                        24C9  7443
                        24C9  7444 IRP_ABORTED:                            ; Abort a CDB IRP
                        24C9  7445          $DISPATCH       IRP$B_XE_FUNC(R3),TYPE=B,-
                        24C9  7446          <-      ;function       action
                        24C9  7447
                        24C9  7448                  <XE_FC_V_LDMEM   10$>,- ; Load DEUNA/DELUA memory
                        24C9  7449                  <XE_FC_V_RDMEM   10$>,- ; Read DEUNA/DELUA memory
                        24C9  7450                  <XE_FC_V_INIT    30$>,- ; Start request
                        24C9  7451                  <XE_FC_V_TESTD   30$>,- ; Self-test done
                        24C9  7452                  <XE_FC_V_RESTART 40$>,- ; Automatic restart
                        24C9  7453                  <XE_FC_V_LDRING  40$>,- ; Load ring pointer
                        24C9  7454                  <XE_FC_V_RDDPA   40$>,- ; Read default physical address
                        24C9  7455                  <XE_FC_V_WTPHYAD 40$>,- ; Write physical address
                        24C9  7456                  <XE_FC_V_WTMULTI 40$>,- ; Write multicast address
                        24C9  7457                  <XE_FC_V_WTMODE  40$>,- ; Write DEUNA/DELUA mode
                        24C9  7458                  <XE_FC_V_START   40$>,- ; Start XMIT/RECV process
                        24C9  7459          >
              14   11   24FE  7460          BRB     20$                    ; All others
                        2500  7461 ;
                        2500  7462 ; For the Load/Read DEUNA/DELUA memory, we must release the MAP registers.
                        2500  7463 ;
     55   00C8 C4   D0  2500  7464 10$:     MOVL    CDB_L_UCB0(R4),R5      ; Get address of UCB unit #0
       52   24 A5   D0  2505  7465          MOVL    UCB$L_CRB(R5),R2       ; Get CRB address
                        2509  7466 ;
                        2509  7467 ;       The read memory and load memory functions do not use the buffered
                        2509  7468 ;       data paths, therefore no purge or release is necessary.
                        2509  7469 ;
                        2509  7470          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                        2509  7471          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
              3C A3   D0  2509  7472          MOVL    IRP$L_XE_MAP(R3),-     ; Setup map register data
              34 A2       250C  7473                   CRB$L_INTD+VEC$W_MAPREG(R2) ;  ...
                        250E  7474          RELMPR                         ; Release map registers
               FA1D   30  2514  7475 20$:     BSBW    ABORT_PKT              ; ABORT the I/O request
                 20   11  2517  7476          BRB     60$                    ; Leave
                        2519  7477 ;
                        2519  7478 ; Self-test failure codes go here
                        2519  7479 ;
```

```
                 0F   E1  2519  7480 30$:    BBC     #XE_PCSR1_V_XPWR,-      ; Br if the XCVR power is okay
          0D 12 A4        251B  7481                 CDB_L_FR3+2(R4),40$     ;  ..must be broken DEUNA/DELUA
     50    01A4 8F   3C  251E  7482          MOVZWL  #SS$_MEDOFL,R0          ; Else, return the correct indicator
                     2523  7483          SETBIT  #XM$V_STS_DISC,CDB_L_DEVDEPEND(R4) ; Save Internal error reason
                 05  11  2529  7484          BRB     50$                     ; Complete the request
                     252B  7485 ;
                     252B  7486 ; All failures during intialization go here
                     252B  7487 ;
     50    0054 8F   3C  252B  7488 40$:    MOVZWL  #SS$_CTRLERR,R0         ; Return broken hardware
00C4 C4      13 A4   90  2530  7489 50$:    MOVB    CDB_L_FR3+3(R4),CDB_L_DEVDEPEND(R4) ; Copy self-test status code
              F9FE   30  2536  7490          BSBW    IO_DONE3                ; Complete the I/O request
                     05  2539  7491 60$:    RSB                             ; Return to caller
```

```
                        253A  7493          .SBTTL SHUTDOWN - SHUT DOWN UNIT
                        253A  7494          .SBTTL SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
                        253A  7495 ;++
                        253A  7496 ; SHUTDOWN - SHUT DOWN UNIT
                        253A  7497 ; SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
                        253A  7498 ;
                        253A  7499 ; Functional description:
                        253A  7500 ;
                        253A  7501 ; This routine is used to shut down the XE unit as a result of a
                        253A  7502 ; SETMODE and SHUTDOWN. The action is to abort all I/O for the unit
                        253A  7503 ; and then to clean up the unit data base.
                        253A  7504 ;
                        253A  7505 ; Inputs:
                        253A  7506 ;
                        253A  7507 ;       R3 = IRP address (SHUTDOWN_PROTYP entry only)
                        253A  7508 ;       R4 = CDB address
                        253A  7509 ;       R5 = UCB address
                        253A  7510 ;
                        253A  7511 ;       IPL = FIPL
                        253A  7512 ;
                        253A  7513 ; Outputs:
                        253A  7514 ;
                        253A  7515 ;       R3-R5 are preserved
                        253A  7516 ;
                        253A  7517 ;       R0-R2 are destroyed.
                        253A  7518 ;--
                        253A  7519 SHUTDOWN_PROTYP::                    ; Shut down protocol type
          04    E1      253A  7520          BBC     #UCB$V_ONLINE,-     ; Br if not online
19 64 A5                253C  7521                  UCB$W_STS(R5),10$   ;
                        253F  7522          ASSUME  UCB$V_XE_INITED  EQ  0
15 68 A5    E9          253F  7523          BLBC    UCB$W_DEVSTS(R5),10$    ; Br if not inited
          03    E1      2543  7524          BBC     #UCB$V_XE_SHARE,-       ; Br if not a shared UCB
11 68 A5                2545  7525                  UCB$W_DEVSTS(R5),SHUTDOWN ; shutdown entire unit
                        2548  7526 ;
                        2548  7527 ; Try to find SHR data structure
                        2548  7528 ;
        E0E2    30      2548  7529          BSBW    MATCH_SHR               ; Check PID and CHAN
          08    12      254B  7530          BNEQ    10$                     ; Br if NO MATCH, skip it
                        254D  7531 ;
                        254D  7532 ; Match found - clear inited bit and clean up all I/O on SHR data structure
                        254D  7533 ;
          56    DD      254D  7534          PUSHL   R6                      ; Save R6
    56    51    D0      254F  7535          MOVL    R1,R6                   ; Copy SHR address
        0296    30      2552  7536          BSBW    CLEANUP_SHR             ; Cleanup the SHR data structure
          56 8ED0       2555  7537          POPL    R6                      ; Restore R6
             05         2558  7538 10$:     RSB                             ; Return to caller
                        2559  7539
                        2559  7540 SHUTDOWN::                             ; Shut down unit
          04    E1      2559  7541          BBC     #UCB$V_ONLINE,-
04 64 A5                255B  7542                  UCB$W_STS(R5),5$        ; If BC not online
                        255E  7543
                        255E  7544          ASSUME  UCB$V_XE_INITED  EQ  0
01 68 A5    E8          255E  7545          BLBS    UCB$W_DEVSTS(R5),10$    ; Br if UCB is inited
             05         2562  7546 5$:      RSB                             ; It's not time to shut down, yet
                        2563  7547
                        2563  7548 ;
                        2563  7549 ; If a power failure occurred, and the protocol has both initialized the FFI
```

```
                                2563  7550 ; interface and supplied an asynchronous error routine, then call back
                                2563  7551 ; the protocol at this routine address with a status value indicating that
                                2563  7552 ; a power failure had taken place.
                                2563  7553 ;
                                2563  7554
                     07   E1    2563  7555 10$:    BBC     #CDB_STS_V_POWER,-      ; Skip notification if power failure
          19 025A    C4         2565  7556                CDB_B_STS(R4),20$       ; did not occur
                     54   DD    2569  7557         PUSHL   R4                      ; Save CDB address
     54   0178    C5   D0    256B  7558         MOVL    UCB$L_XE_FFI(R5),R4     ; Retrieve FFI address
                     0D   13    2570  7559         BEQL    15$                     ; Nothing to do if there isn't one
       52    1C    A4   D0    2572  7560         MOVL    FFI$L_ERROR(R4),R2      ; Retrieve asynch error routine address
                     07   13    2576  7561         BEQL    15$                     ; Nothing to do if there isn't one
     50   C364    8F   3C    2578  7562         MOVZWL  #SS$_POWERFAIL,R0       ; Indicate that a powerfailure occurred
                     62   16    257D  7563         JSB     (R2)                    ; Call back the asynch error routine
                     54 8ED0    257F  7564 15$:    POPL    R4                      ; Restore CDB address
                                2582  7565
                                2582  7566 ;
                                2582  7567 ; Start a 3-second timer to restart any UNIT needing automatic restart.
                                2582  7568 ; This restart timer only runs if the device was halted due to a fatal error.
                                2582  7569 ;
                                2582  7570 ; Note, that the UCB multicast address list is purged, which will nullify
                                2582  7571 ; any restart operation that may be performed by the DEUNA/DELUA driver itself
                                2582  7572 ; (only if the user has specified any multicast addresses).
                                2582  7573 ;
                                2582  7574
                   00C8 8F   BB    2582  7575 20$:    PUSHR   #^M<R3,R6,R7>           ; Save registers
                     0F   E1    2586  7576         BBC     #UCB$V_XE_RESTART,-     ; Br if this UNIT does not need
                  43 68 A5         2588  7577                 UCB$W_DEVSTS(R5),22$    ;  automatic restart
                                258B  7578         ASSUME  IRP$C_LENGTH GE TQE$C_LENGTH
       51    F4 8F   9A    258B  7579         MOVZBL  #IRP$C_LENGTH+TQE$C_LENGTH,R1 ; Get size of IRP/TQE
 00000000'GF   16    258F  7580         JSB     G^EXE$ALONONPAGED       ; Try to allocate a IRP/TQE
         36 50   E9    2595  7581         BLBC    R0,22$                  ; Br if failure - too bad
                                2598  7582         ASSUME  IRP$Q_STATION GT TQE$C_LENGTH
             4000 8F   A8    2598  7583         BISW    #UCB$M_XE_INTERLOCK,-   ; Interlock the RESTART operation
               68 A5         259C  7584                 UCB$W_DEVSTS(R5)        ;
                                259E  7585
                                259E  7586         ASSUME  TQE$B_TYPE EQ TQE$W_SIZE+2
                                259E  7587         ASSUME  TQE$B_RQTYPE EQ TQE$B_TYPE+1
          000F00F4 8F   D0    259E  7588         MOVL    #<<DYN$C_TQE@16>!<IRP$C_LENGTH+TQE$C_LENGTH>>,- ; Set STRUCTURE
                 08 A2         25A4  7589                 TQE$W_SIZE(R2)          ;  TYPE and SIZE
             00C8 C2   55   D0    25A6  7590         MOVL    R5,TQE$C_LENGTH+IRP$L_RBOFF(R2) ; Save UCB address in IRP
                                25AB  7591         PUSHQ   R4                      ; Save R4, R5
                   55   52   D0    25AE  7592         MOVL    R2,R5                   ; Copy TQE address
00000000 01C9C380 8F   7D    25B1  7593         MOVQ    #RESTART_DELTA,-        ; Set the delta time
                 20 A5         25BB  7594                 TQE$Q_DELTA(R5)         ;
          53   FC74 CF   9E    25BD  7595         MOVAB   W^RESTART_ROUT,R3       ; Get address of RESTART routine
                   01   90    25C2  7596         MOVB    #TQE$C_SSSNGL,-         ; Set the request type
                 2C A5         25C4  7597                 TQE$L_RQPID(R5)         ;
                   ECF8 30    25C6  7598         BSBW    FORK_TIMER              ; FORK to startup the timer
                                25C9  7599         POPQ    R4                      ; Restore R4, R5
                   05   11    25CC  7600         BRB     23$                     ; Continue
                                25CE  7601
                                25CE  7602 22$:    CLRBIT  #UCB$V_XE_RESTART,-     ; Restart is not possible!
                                25CE  7603                  UCB$W_DEVSTS(R5)
   00D2 C5   34BC 8F   B0    25D3  7604 23$:    MOVW    #INIT_Q_QUOTA,UCB$W_XE_HBQ(R5) ; Reset hardware buffer quota
         64 A5   23   AA    25DA  7605         BICW    #UCB$M_INT!UCB$M_POWER!-
                                25DE  7606                 UCB$M_TIM,UCB$W_STS(R5) ; Reset device status
```

```
                68 A5    11    AA  25DE  7607          BICW    #UCB$M_XE_INITED!-      ; No longer inited
                                   25E2  7608                   UCB$M_XE_RUN,UCB$W_DEVSTS(R5) ; ..or  running
                                   25E2  7609          CLRBIT  #XM$V_STS_ACTIVE,UCB$L_DEVDEPEND(R5) ; Clear active bit
             05 00C4 C4    10    E1  25E7  7610          BBC     #XM$V_ERR_FATAL,CDB_L_DEVDEPEND(R4),25$ ; Br if not FATAL
                                   25ED  7611          SETBIT  #XM$V_ERR_FATAL,UCB$L_DEVDEPEND(R5) ; Else, indicate FATAL
                                   25F2  7612 ;
                                   25F2  7613 ; Reset UCB multicast address list
                                   25F2  7614 ;
              0224 C4    55    D1  25F2  7615 25$:     CMPL    R5,CDB_L_PRMUSER(R4)    ; Is this unit the PROMISCUOUS user?
                         0D    12  25F7  7616          BNEQ    27$                     ; Br if not
                   0224 C4    D4  25F9  7617          CLRL    CDB_L_PRMUSER(R4)       ; Else, clear the PROMISCUOUS user addr
                                   25FD  7618          CLRBIT  #PB_MOD_V_PROM,CDB_W_MODE(R4) ; Reset from promiscuous mode
                      0146    30  2603  7619          BSBW    BLD_STOP_IRP            ; Build an IRP to RESET hardware mode
                        0F    E0  2606  7620 27$:     BBS     #UCB$V_XE_RESTART,-     ; Br if this UNIT is restarting
                   14 68 A5         2608  7621                  UCB$W_DEVSTS(R5),28$    ;  don't clear multicast list
                                   260B  7622          PUSHQ   R4                      ; Save R4, R5
                   00E7 C5    94    260E  7623          CLRB    UCB$B_XE_MULTI(R5)      ; No more multicast addresses
          00       00E9 C5    00    2C  2612  7624          MOVC5   #0,UCB$G_XE_MULTI(R5),#0,- ; Zero the structure
                   00E9 C5    3C        2618  7625                  #6*MAX_C_MLT,UCB$G_XE_MULTI(R5) ;
                                   261C  7626          POPQ    R4                      ; Restore R4, R5
                                   261F  7627 ;
                                   261F  7628 ; Reset CDB multicast address list, Flush all attention ASTs.
                                   261F  7629 ;
                      04AB    30  261F  7630 28$:     BSBW    ADD_MULTI               ; Re-calculate multicast address list
                        54    DD  2622  7631          PUSHL   R4                      ; Save CDB address
                57   00C0 C5    9E  2624  7632 30$:     MOVAB   UCB$L_XE_AST(R5),R7     ; Get address of AST listhead
                     50   67    D0  2629  7633          MOVL    (R7),R0                 ; Anything in list?
                          1B    13  262C  7634          BEQL    40$                     ; Br if not
                   56   22 A0    3C  262E  7635          MOVZWL  ACB$L_KAST+10(R0),R6    ; Force channel match
                   52   24 A0    3C  2632  7636          MOVZWL  ACB$L_KAST+12(R0),R2    ; Get process index
             54   00000000'GF    D0  2636  7637          MOVL    G^SCH$GL_PCBVEC,R4      ; Get PCB address vector address
                     54   6442    D0  263D  7638          MOVL    (R4)[R2],R4             ; Get PCB address
                   00000000'GF    16  2641  7639          JSB     G^COM$FLUSHATTNS        ; Flush AST
                                   DB  2647  7640          BRB     30$
                                   54 8ED0  2649  7641 40$:     POPL    R4                      ; Restore CDB address
                                            264C  7642 ;
                                            264C  7643 ; Complete all RCV IRPs for this unit
                                            264C  7644 ;
                                            264C  7645          ASSUME  UCB$C_XE_QUEUES-1 EQ 3  ; One queue for shared users
          53   00A8 D5    0F    264C  7646 45$:     REMQUE  @UCB$Q_XE_RCVREQ(R5),R3 ; Get IRP
                         05    1D  2651  7647          BVS     50$                     ; Br if none
                      F8DE    30  2653  7648          BSBW    ABORT_PKT               ; Abort the I/O request
                        F4    11  2656  7649          BRB     45$                     ; Get next IRP
                                   2658  7650 ;
                                   2658  7651 ; Complete all XMIT CXBs for this unit
                                   2658  7652 ;
          53   00B0 D5    0F    2658  7653 50$:     REMQUE  @UCB$Q_XE_XMTREQ(R5),R3 ; Get CXB
                         20    1D  265D  7654          BVS     55$                     ; Br if none
                50   24 A3    D0  265F  7655          MOVL    CXB$L_T_IRP(R3),R0      ; Get IRP address
                         12    12  2663  7656          BNEQ    54$                     ; Br if present
                         54    DD  2665  7657          PUSHL   R4                      ; Save CDB address
          54   0178 C5    D0    2667  7658          MOVL    UCB$L_XE_FFI(R5),R4     ; Get FFI block address
                     50   2C    3C  266C  7659          MOVZWL  #SS$_ABORT,R0           ; Set status return
                        14 B4    16  266F  7660          JSB     @FFI$L_XMIT_DONE(R4)    ; Complete CXB
                                   54 8ED0  2672  7661          POPL    R4                      ; Restore CDB address
                                   E1  11  2675  7662          BRB     50$                     ; Get next CXB
                                           2677  7663
```

```
                53   50    D0  2677  7664 54$:    MOVL    R0,R3                   ; Copy IRP address
                   F8B7    30  267A  7665         BSBW    ABORT_PKT               ; Abort the I/O request
                     D9    11  267D  7666         BRB     50$                     ; Get next CXB
                           267F  7667 ;
                           267F  7668 ; Deallocate all receive CXBs
                           267F  7669 ;
            52   00A0 D5    0F  267F  7670 55$:    REMQUE  @UCB$Q_XE_RCVMSG(R5),R2 ; Get message buffer
                     24    1D  2684  7671         BVS     70$                     ; Br if none
                   42 A5   A0  2686  7672         ADDW    UCB$W_DEVBUFSIZ(R5),-   ; Restore quota
                00C8 C5        2689  7673                 UCB$W_XE_QUOTA(R5)      ;
                           268C  7674 ;
                           268C  7675 ;       The buffer may be smaller than the normal message size, if this
                           268C  7676 ;       is a cloned buffer for the promiscuous user. Therefore, we must
                           268C  7677 ;       check to make sure the buffer is large enough to be returned
                           268C  7678 ;       the the device's receive buffer pool.
                           268C  7679 ;
                     A1    268C  7680         ADDW3   #CXB$C_HEADER+-         ; Calculate size of "normal"
                           268D  7681                 CXB$C_TRAILER,-         ;  receive buffer
        50   00B8 C4   004C 8F    268D  7682                 CDB_W_BSZ(R4),R0        ;
                08 A2    50   B1  2694  7683         CMPW    R0,CXB$W_SIZE(R2)       ; Can buffer be returned?
                     05    12  2698  7684         BNEQ    60$                     ; Br if not, delete buffer instead
                   ED12    30  269A  7685         BSBW    ADDRCVLIST              ; Try to add to receiver list
                     B9    11  269D  7686         BRB     50$                     ; Loop for more
                           269F  7687
                     50   52   D0  269F  7688 60$:    MOVL    R2,R0                   ; Copy buffer address for deallocation
             00000000'GF    16  26A2  7689         JSB     G^COM$DRVDEALMEM        ; Deallocate the buffer
                     AE    11  26A8  7690         BRB     50$                     ; Loop for more
                           26AA  7691 ;
                           26AA  7692 ; Cleanup all SHR structures if fatal error
                           26AA  7693 ;
                     03    E1  26AA  7694 70$:    BBC     #UCB$V_XE_SHARE,-       ; Br if not a SHARED UCB
                30 68 A5       26AC  7695                 UCB$W_DEVSTS(R5),100$   ;
            56   00C4 C5   D0  26AF  7696         MOVL    UCB$L_XE_DEFUSR(R5),R6  ; Get default SHR structure address
                     0B    13  26B4  7697         BEQL    90$                     ; Br if none
                   0132    30  26B6  7698         BSBW    CLEANUP_SHR             ; Else, cleanup the structure
                     10    E1  26B9  7699         BBC     #XM$V_ERR_FATAL,-       ; Br if not a fatal error
                03 44 A5       26BB  7700                 UCB$L_DEVDEPEND(R5),90$
                   0178    30  26BE  7701         BSBW    DELETE_SHR              ; And delete the structure
            56   0098 C5   D0  26C1  7702 90$:    MOVL    UCB$Q_XE_SHARE(R5),R6   ; Get address of next LIMITED user
        0098 C5     56   D1  26C6  7703         CMPL    R6,UCB$Q_XE_SHARE(R5)   ; End of list?
                     45    13  26CB  7704         BEQL    120$                    ; Br if yes, don't restore quota (yet)
                   011B    30  26CD  7705         BSBW    CLEANUP_SHR             ; Cleanup the I/O
                     10    E1  26D0  7706         BBC     #XM$V_ERR_FATAL,-       ; Br if not a fatal error
                EC 44 A5       26D2  7707                 UCB$L_DEVDEPEND(R5),90$
                     0F    E0  26D5  7708         BBS     #UCB$V_XE_RESTART,-     ; Br if this UNIT is re-starting
                E7 68 A5       26D7  7709                 UCB$W_DEVSTS(R5),90$
                   015C    30  26DA  7710         BSBW    DELETE_SHR              ; Else, delete the structure
                     E2    11  26DD  7711         BRB     90$                     ; Look for more
                           26DF  7712 ;
                           26DF  7713 ; Restore quota
                           26DF  7714 ;
                     0F    E0  26DF  7715 100$:   BBS     #UCB$V_XE_RESTART,-     ; Br if this UNIT is re-starting
                5A 68 A5       26E1  7716                 UCB$W_DEVSTS(R5),140$
            50   00B8 C5   3C  26E4  7717         MOVZWL  UCB$L_XE_PID(R5),R0     ; Get PID of last starter
        51   00000000'GF    D0  26E9  7718         MOVL    G^SCH$GL_PCBVEC,R1      ; Address PCB vector
               50   6140    D0  26F0  7719         MOVL    (R1)[R0],R0             ; Get PCB of owner
                   60 A0    D1  26F4  7720         CMPL    PCB$L_PID(R0),-
```

```
                00B8 C5      26F7  7721                    UCB$L_XE_PID(R5)        ; Still there?
                     16  12  26FA  7722            BNEQ    120$                    ; If NEQ no
          50    0080 C0  D0  26FC  7723            MOVL    PCB$L_JIB(R0),R0        ; Get JIB address
          51    00C8 C5  3C  2701  7724            MOVZWL  UCB$W_XE_QUOTA(R5),R1   ; Convert to longword
            20  A0   51  C0  2706  7725            ADDL    R1,JIB$L_BYTCNT(R0)     ; Return byte count quota
            24  A0   51  C0  270A  7726            ADDL    R1,JIB$L_BYTLM(R0)      ; ..and byte limit quota
                00C8 C5  B4  270E  7727            CLRW    UCB$W_XE_QUOTA(R5)      ; Prevent this from being
                             2712  7728                                            ; returned again
                             2712  7729 ;
                             2712  7730 ; Delete the STARTUP IRP for point-to-point mode
                             2712  7731 ;
          50    017C C5  D0  2712  7732 120$:  MOVL    UCB$L_XE_STIRP(R5),R0   ;% Get the startup IRP address
                     0A  13  2717  7733            BEQL    130$                    ;% Br if none
                017C C5  D4  2719  7734            CLRL    UCB$L_XE_STIRP(R5)      ;% All done
           00000000'GF   16  271D  7735            JSB     G^COM$DRVDEALMEM        ;% Deallocate the IRP
                             2723  7736 ;
                             2723  7737 ; If there is an FFI block and the SHUT_DONE routine is set, then
                             2723  7738 ; notify the FFI user that shutdown is now complete.
                             2723  7739 ;
          50    0178 C5  D0  2723  7740 130$:  MOVL    UCB$L_XE_FFI(R5),R0     ; Get FFI block address
                     14  13  2728  7741            BEQL    140$                    ; Br if none
                0178 C5  D4  272A  7742            CLRL    UCB$L_XE_FFI(R5)        ; Cleanup FFI interface
            51    20 A0  D0  272E  7743            MOVL    FFI$L_SHUT_DONE(R0),R1  ; Get address of routine
                     0A  13  2732  7744            BEQL    140$                    ; Br if none
                     54  DD  2734  7745            PUSHL   R4                      ; Save CDB address
                54   50  D0  2736  7746            MOVL    R0,R4                   ; Copy FFI block address
                     61  16  2739  7747            JSB     (R1)                    ; Call back FFI user
                     54 8ED0 273B  7748            POPL    R4                      ; Restore CDB address
                             273E  7749 ;
                             273E  7750 ; Decrement UNIT count on CDB and cleanup CDB if last unit
                             273E  7751 ;
                0275 C4  97  273E  7752 140$:  DECB    CDB_B_UNTCNT(R4)        ; One less unit on CDB
                     03  12  2742  7753            BNEQ    150$                    ; Br if more
                   FC81  30  2744  7754            BSBW    SHUTDOWN_UNA            ; Else, shutdown entire DEUNA/DELUA
           00C8 8F     BA  2747  7755 150$:  POPR    #^M<R3,R6,R7>           ; Restore registers
                     05  274B  7756            RSB                             ; Return to caller
                             274C  7757
```

```
                                274C  7759          .SBTTL  BLD_STOP_IRP - Build an IRP to reset promiscuous mode
                                274C  7760 ;++
                                274C  7761 ; BLD_STOP_IRP - Build an IRP to reset the promiscuous mode
                                274C  7762 ;
                                274C  7763 ; Functional description:
                                274C  7764 ;
                                274C  7765 ; This routine will allocate and build an IRP to reset the hardware mode
                                274C  7766 ; from promiscous.
                                274C  7767 ;
                                274C  7768 ; Inputs:
                                274C  7769 ;       R4 = CDB address
                                274C  7770 ;       R5 = UCB address
                                274C  7771 ;
                                274C  7772 ; Outputs:
                                274C  7773 ;       R0,R1,R2,R3 are destroyed.
                                274C  7774 ;--
                                274C  7775 BLD_STOP_IRP:                            ; Build an IRP to reset hardware mode
                                274C  7776 ;
                                274C  7777 ; NOTE - we must use EXE$ALONONPAGED to allocate the IRP because the other
                                274C  7778 ;       routines reset the IPL to ASTDEL.
                                274C  7779 ;
            10   E0  274C  7780          BBS     #XM$V_ERR_FATAL,-          ; Br if fatal error,
 0E 00C4 C4         274E  7781                  CDB_L_DEVDEPEND(R4),10$ ;  ignore reset of mode
51   00C4 8F   3C  2752  7782          MOVZWL  #IRP$C_LENGTH,R1           ; Set length of IRP
 00000000'GF   16  2757  7783          JSB     G^EXE$ALONONPAGED          ; Try to allocate an IRP
       01 50   E8  275D  7784          BLBS    R0,20$                     ; Okay if buffer allocated
               05  2760  7785 10$:     RSB                                ; Else, too bad if we can't do it
                   2761  7786
  08 A2   51   B0  2761  7787 20$:     MOVW    R1,IRP$W_SIZE(R2)          ; Fill in the size field
               67  10  2765  7788          BSBB    BLD_IRP_CO                 ; Build a template IRP
0C A3    71'AF   9E  2767  7789          MOVAB   B^DELETE_BLOCK,IRP$L_PID(R3) ; Store return address from IOPOST
  21 A3    09   90  276C  7790          MOVB    #XE_FC_V_STOP,IRP$B_XE_FUNC(R3) ; Make this look like a STOP
               05  2770  7791          RSB                                ; Return to queue request to DEUNA/DELUA
                   2771  7792
                   2771  7793 DELETE_BLOCK:                                ; Deallocate a data structure
    50   55   D0  2771  7794          MOVL    R5,R0                      ; Get address of structure
 00000000'GF   17  2774  7795          JMP     G^COM$DRVDEALMEM           ; Deallocate the structure
                   277A  7796
```

```
                                277A  7798          .SBTTL  BLD_STRT_IRP - Build a point-to-point startup IRP
                                277A  7799 ;++
                                277A  7800 ; BLD_STRT_IRP - Build a point-to-point startup IRP
                                277A  7801 ;
                                277A  7802 ; Functional description:
                                277A  7803 ;
                                277A  7804 ; This routine will build an IRP to perform a datalink startup with a
                                277A  7805 ; remote system.
                                277A  7806 ;
                                277A  7807 ; Inputs:
                                277A  7808 ;       R2 = IRP address
                                277A  7809 ;       R4 = CDB address
                                277A  7810 ;       R5 = UCB address
                                277A  7811 ;
                                277A  7812 ; Outputs:
                                277A  7813 ;       R0,R1,R2,R3 are destroyed.
                                277A  7814 ;--
                                277A  7815 BLD_STRT_IRP:                            ;% Build a startup IRP
                                277A  7816 ;
                                277A  7817 ; We will use the back part of the IRP to build the data message. The
                                277A  7818 ; data message only contains the standard header plus one byte of ^XAA.
                                277A  7819 ;
                                277A  7820          ASSUME  IRP$C_XE_STD+17 LE IRP$C_LENGTH ;% 14 bytes of header +
                                277A  7821                                              ;%    2 bytes of count + msg type.
                          3C 10 277A  7822          BSBB    BLD_IRP                   ;% Build the IRP
                       52 60 A3 9E 277C  7823          MOVAB   IRP$C_XE_STD(R3),R2       ;% Point to data portion of IRP
                          00CC C5 7D 2780  7824          MOVQ    UCB$G_XE_DES(R5),-        ;% Store destination address
                                62 2784  7825                  XBUF_G_DEST(R2)           ;%
                   0C A2    0660 8F B0 2785  7826          MOVW    #XE_C_STPRO,XBUF_W_TYPE(R2) ;% Store protocol type
                       0E A2    01 B0 278B  7827          MOVW    #1,XBUF_W_SIZE(R2)        ;% Store message size
                     10 A2    AA 8F 90 278F  7828          MOVB    #^XAA,XBUF_W_SIZE+2(R2) ;% Transmit one start byte of data
                     05 68 A5    05 E0 2794  7829          BBS     #UCB$V_XE_START,UCB$W_DEVSTS(R5),50$ ;% Br if start
                                   2799  7830 ;;&&       BBC     #UCB$V_XE_STACK,UCB$W_DEVSTS(R5),90$ ;% Exit if not s   k
                     10 A2    AB 8F 90 2799  7831          MOVB    #^XAB,XBUF_W_SIZE+2(R2) ;% Transmit one stack byte of   ta
                     0C A3   B7'AF 9E 279E  7832 50$:     MOVAB   B^90$,IRP$C_PID(R3)       ;% Store return address
                        21 A3    07 90 27A3  7833          MOVB    #XE_FC_V_XMIT,IRP$B_XE_FUNC(R3) ;% Set function request
                                   27A7  7834          ASSUME  XBUF_C_HEADER_EQ XBUF_W_SIZE
                                11 B0 27A7  7835          MOVW    #XBUF_C_HEADER+3,-        ;% Set data size
                             32 A3    27A9  7836                  IRP$W_BCNT(R3)            ;%
                          3C A3    52 D0 27AB  7837          MOVL    R2,IRP$L_XE_SYSBUF(R3)    ;% Set buffer address
                       0080 D4    63 0E 27AF  7838          INSQUE  (R3),@CDB_Q_XMTREQ+4(R4);% Insert transmit request
                                DCF1 30 27B4  7839          BSBW    XMT_ALT_START             ;% Startup transmit process
                                   05 27B7  7840 90$:     RSB                               ;% Return to caller
                                      27B8  7841
```

```
                     27B8  7843          .SBTTL  BLD_IRP - Build an IRP routine
                     27B8  7844          .SBTTL  BLD_IRP_CO - Build an IRP co-routine
                     27B8  7845 ;++
                     27B8  7846 ; BLD_IRP - Build an IRP routine
                     27B8  7847 ; BLD_IRP_CO - Build an IRP co-routine
                     27B8  7848 ;
                     27B8  7849 ; Functional description:
                     27B8  7850 ;
                     27B8  7851 ; This routine will build a simple IRP and allow the caller to fill in the
                     27B8  7852 ; function requested and then queue it to the DEUNA/DELUA.
                     27B8  7853 ;
                     27B8  7854 ; Inputs:
                     27B8  7855 ;       R2 = IRP address
                     27B8  7856 ;       R5 = UCB address
                     27B8  7857 ;
                     27B8  7858 ; Outputs:
                     27B8  7859 ;       R0-R3 are destroyed.
                     27B8  7860 ;       R4,R5 are preserved.
                     27B8  7861 ;--
                     27B8  7862 BLD_IRP:                                ; Build an IRP
53   82   7E           27B8  7863          MOVAQ   (R2)+,R3               ; Save IRP address, skip to size field
                     27BB  7864          ASSUME  IRP$W_SIZE EQ 8
                     27BB  7865          ASSUME  IRP$B_TYPE EQ IRP$W_SIZE+2
                     27BB  7866          ASSUME  IRP$B_RMOD EQ IRP$B_TYPE+1
     82   B5         27BB  7867          TSTW    (R2)+                  ; Skip SIZE
82   0A   B0         27BD  7868          MOVW    #DYN$C_IRP,(R2)+       ; Make it look like an IRP
                     27C0  7869          ASSUME  IRP$L_PID EQ IRP$B_RMOD+1
                     27C0  7870          ASSUME  IRP$L_AST EQ IRP$L_PID+4
     82   7C         27C0  7871          CLRQ    (R2)+                  ; Clear PID, AST
                     27C2  7872          ASSUME  IRP$L_ASTPRM EQ IRP$L_AST+4
                     27C2  7873          ASSUME  IRP$L_WIND EQ IRP$L_ASTPRM+4
     82   7C         27C2  7874          CLRQ    (R2)+                  ; Clear ASTPRM, WIND
                     27C4  7875          ASSUME  IRP$L_UCB EQ IRP$L_WIND+4
82   55   D0         27C4  7876          MOVL    R5,(R2)+               ; Store UCB address
                     27C7  7877          ASSUME  IRP$W_FUNC EQ IRP$L_UCB+4
                     27C7  7878          ASSUME  IRP$B_EFN EQ IRP$W_FUNC+2
                     27C7  7879          ASSUME  IRP$B_PRI EQ IRP$B_EFN+1
                     27C7  7880          ASSUME  IRP$L_IOSB EQ IRP$B_PRI+1
     82   7C         27C7  7881          CLRQ    (R2)+                  ; Clear FUNC, EFN, PRI, IOSB
                     27C9  7882          ASSUME  IRP$W_CHAN EQ IRP$L_IOSB+4
                     27C9  7883          ASSUME  IRP$W_STS EQ IRP$W_CHAN+2
                     27C9  7884          ASSUME  IRP$L_SVAPTE EQ IRP$W_STS+2
     82   7C         27C9  7885          CLRQ    (R2)+                  ; Clear CHAN, STS, SVAPTE
                     27CB  7886          ASSUME  IRP$W_BOFF EQ IRP$L_SVAPTE+4
                     27CB  7887          ASSUME  IRP$W_BCNT EQ IRP$W_BOFF+2
                     27CB  7888          ASSUME  IRP$L_BCNT EQ IRP$W_BCNT
     82   7C         27CB  7889          CLRQ    (R2)+                  ; Clear BOFF, BCNT
     05   27CD  7890          RSB                                    ; Return to caller
                     27CE  7891
                     27CE  7892 BLD_IRP_CO:                             ; Build an IRP - co-routine call
     E8   10         27CE  7893          BSBB    BLD_IRP                ; First, build the IRP
                     27D0  7894          ;
                     27D0  7895          ; On return to caller
                     27D0  7896          ;
                     27D0  7897          ;       R3 = IRP address
                     27D0  7898          ;
     9E   16         27D0  7899          JSB     @(SP)+                 ; Return to caller
```

```
                     27D2  7900          ;
                     27D2  7901          ; After call back:
                     27D2  7902          ;
                     27D2  7903          ;       R4 = CDB address
                     27D2  7904          ;       R5 = UCB address
                     27D2  7905          ;       R0-R2 may be destroyed
                     27D2  7906          ;
                     27D2  7907          DSBINT  UCB$B_DIPL(R5)              ; Sync access to device
0088 D4    63    0E  27D9  7908          INSQUE  (R3),@CDB_Q_INPUT+4(R4)     ; Insert at END of queue
                     27DE  7909          PUSHQ   R4                          ; Save CDB, UCB addresses
           ECD8    30  27E1  7910          BSBW    LOAD_PORT                   ; Load request and return to caller
                     27E4  7911          POPQ    R4                          ; Restore CDB, UCB addresses
                     27E7  7912          ENBINT                              ; Re-enable interrupts
                  05 27EA  7913          RSB                                 ; Return to caller
                     27EB  7914
```

```
                     27EB  7916          .SBTTL  CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
                     27EB  7917          .SBTTL  DELETE_SHR - DELETE SHR DATA STRUCTURE
                     27EB  7918 ;++
                     27EB  7919 ; CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
                     27EB  7920 ;
                     27EB  7921 ; This routine aborts all read request in progress and return all message
                     27EB  7922 ; buffers back to the CDB structure for re-use.
                     27EB  7923 ;
                     27EB  7924 ; Inputs:
                     27EB  7925 ;
                     27EB  7926 ;       R4 = CDB address
                     27EB  7927 ;       R5 = UCB address
                     27EB  7928 ;       R6 = SHR address
                     27EB  7929 ;
                     27EB  7930 ;       IPL = FIPL
                     27EB  7931 ;
                     27EB  7932 ; Outputs:
                     27EB  7933 ;
                     27EB  7934 ;       R0-R2 are destroyed.
                     27EB  7935 ;       All other registers are preserved.
                     27EB  7936 ;--
                     27EB  7937 CLEANUP_SHR::                            ; Cleanup all I/O on SHR structure
          53   DD    27EB  7938          PUSHL   R3                      ; Save R3
                     27ED  7939
                     27ED  7940          .IF DF  POINT
          57   DD    27ED  7941          PUSHL   R7                      ;% Save R7
                     27EF  7942 ;
                     27EF  7943 ; Complete all waiting transmit IRPs
                     27EF  7944 ;
    57   00B0 C5  9E 27EF  7945 3$:     MOVAB   UCB$Q_XE_XMTREQ(R5),R7  ;% Get address of XMIT wait queue
         53   67  D0 27F4  7946          MOVL    (R7),R3                 ;% Travel queue
         57   53  D1 27F7  7947 5$:      CMPL    R3,R7                   ;% At end of queue?
              1A  13 27FA  7948          BEQL    9$                      ;% Br if yes
    50    24 A3  D0 27FC  7949          MOVL    CXB$L_T_IRP(R3),R0      ;% Get IRP address
              07  13 2800  7950          BEQL    6$                      ;% Br if none, only one chan allowed
10  A6    28 A0  B1 2802  7951          CMPW    IRP$W_CHAN(R0),SHR_W_CHAN(R6) ;% Same as SHR chan?
              08  12 2807  7952          BNEQ    7$                      ;% Br if not, else
         53   63  0F 2809  7953 6$:      REMQUE  (R3),R3                 ;% Remove IRP from list
            F725  30 280C  7954          BSBW    ABORT_PKT               ;% Abort the I/O request
              DE  11 280F  7955          BRB     3$                      ;% Look for more
         53   63  D0 2811  7956 7$:      MOVL    (R3),R3                 ;% Travel link
              E1  11 2814  7957          BRB     5$                      ;% Check for end of queue
              57 8ED0 2816  7958 9$:      POPL    R7                      ;% Restore R7
                     2819  7959          .ENDC
                     2819  7960
                     2819  7961          ASSUME  SHR_C_QUEUES EQ 2
                     2819  7962 ;
                     2819  7963 ; Complete all IRPs for this structure
                     2819  7964 ;
    53   20 B6    0F 2819  7965 10$:     REMQUE  @SHR_Q_RCVREQ(R6),R3    ; Get IRP
              05  1D 281D  7966          BVS     20$                     ; Br if none
            F712  30 281F  7967          BSBW    ABORT_PKT               ; Abort the I/O request
              F5  11 2822  7968          BRB     10$                     ; Get next IRP
                     2824  7969 ;
                     2824  7970 ; Deallocate all message blocks
                     2824  7971 ;
    52   18 B6    0F 2824  7972 20$:     REMQUE  @SHR_Q_RCVMSG(R6),R2    ; Get message buffer
```

```
                                                J 11
                 0B   1D  2828  7973          BVS     30$                             ; Br if none
               42 A5  A0  282A  7974          ADDW    UCB$W_DEVBUFSIZ(R5),-           ; Restore quota
            00C8 C5       282D  7975                   UCB$W_XE_QUOTA(R5)              ;
               EB7C  30   2830  7976          BSBW    ADDRCVLIST                      ; Try to add to receiver list
                 EF  11   2833  7977          BRB     20$                             ; Loop
                 53 8ED0  2835  7978 30$:      POPL    R3                              ; Restore R3
                    05    2838  7979          RSB                                     ; Return to caller
                          2839  7980
                          2839  7981 ;++
                          2839  7982 ; DELETE_SHR - DELETE SHARE DATA STRUCTURE
                          2839  7983 ;
                          2839  7984 ; This routine deallocates the SHR data structure to system pool.
                          2839  7985 ;
                          2839  7986 ; Inputs:
                          2839  7987 ;
                          2839  7988 ;       R5 = UCB address
                          2839  7989 ;       R6 = SHR address
                          2839  7990 ;
                          2839  7991 ;       IPL = FIPL
                          2839  7992 ;
                          2839  7993 ; Outputs:
                          2839  7994 ;
                          2839  7995 ;       R0-R1 are destroyed.
                          2839  7996 ;       All other registers are preserved.
                          2839  7997 ;--
                          2839  7998 DELETE_SHR::                                    ; Delete SHR data structure
                    5C A5 B7    2839  7999          DECW    UCB$W_REFC(R5)          ; One less user of the unit
            00C4 C5     56 D1   283C  8000          CMPL    R6,UCB$L_XE_DEFUSR(R5)  ; Is this the default user?
                        1C 13   2841  8001          BEQL    30$                     ; Br if yes
            51  0098 C5  9E   2843  8002          MOVAB   UCB$Q_XE_SHARE(R5),R1   ; Get address of SHARE queue
                50 61   D0   2848  8003          MOVL    (R1),R0                 ; Get address of next in queue
                51 50   D1   284B  8004 10$:      CMPL    R0,R1                   ; Back to front of list?
                        4D 13   284E  8005          BEQL    90$                     ; Br if none found
                50      56 D1   2850  8006          CMPL    R6,R0                   ; Is this the one?
                        05 13   2853  8007          BEQL    20$                     ; Br if yes
                50      60 D0   2855  8008          MOVL    (R0),R0                 ; Else, get next in queue
                        F1 11   2858  8009          BRB     10$                     ; And try for match
                50      60 0F   285A  8010 20$:      REMQUE  (R0),R0                 ; Remove structure from list
                        04 11   285D  8011          BRB     40$                     ; And delete the structure
                   00C4 C5  D4  285F  8012 30$:      CLRL    UCB$L_XE_DEFUSR(R5)     ; No more default user
                 50    0C A6  3C  2863  8013 40$:      MOVZWL  SHR_L_PID(R6),R0        ; Get PID SHR structure
            51    00000000'GF  D0  2867  8014          MOVL    G^SCH$GL_PCBVEC,R1      ; Address PCB vector
                  50   6140  D0  286E  8015          MOVL    (R1)[R0],R0             ; Get PCB of owner
                       60 A0  D1  2872  8016          CMPL    PCB$L_PID(R0),-
                       0C A6      2875  8017                  SHR_L_PID(R6)           ; Still there?
                          1B  12  2877  8018          BNEQ    60$                     ; If NEQ no
                50   0080 C0  D0  2879  8019          MOVL    PCB$L_JIB(R0),R0        ; Get JIB address
                  51    28 A6  3C  287E  8020          MOVZWL  SHR_W_QUOTA(R6),R1      ; Convert to longword
                  20 A0    >1  C0  2882  8021          ADDL    R1,JIB$L_BYTCNT(R0)     ; Return byte count quota
                  24 A0    51  C0  2886  8022          ADDL    R1,JIB$L_BYTLM(R0)      ; ..and byte limit quota
               00C8 C5     51  A2  288A  8023          SUBW    R1,UCB$W_XE_QUOTA(R5)   ; Decrease the current quota
               0175 C5     51  A2  288F  8024          SUBW    R1,UCB$W_XE_TOTQUO(R5)  ;  and the total quota
                    50     56  D0  2894  8025 60$:      MOVL    R6,R0                   ; Copy SHR structure address
                 00000000'GF  17  2897  8026          JMP     G^COM$DRVDEALMEM        ; Deallocate the structure
                              289D  8027
                              289D  8028 ;
                              289D  8029 ; Bug check on error
```

```
289D  8030 ;
289D  8031 90$:    BUG_CHECK NOBUFPCKT,FATAL
```

```
                     28A1  8033          .SBTTL  CANCEL - CANCEL I/O ON UNIT
                     28A1  8034 ;++
                     28A1  8035 ; CANCEL - CANCEL I/O ON UNIT
                     28A1  8036 ;
                     28A1  8037 ; Functional description:
                     28A1  8038 ;
                     28A1  8039 ; This routine is used to cancel specific or all I/O pending on an XE unit.
                     28A1  8040 ;
                     28A1  8041 ; Inputs:
                     28A1  8042 ;
                     28A1  8043 ;       R2 = Channel index number
                     28A1  8044 ;       R4 = PCB address (or zero)
                     28A1  8045 ;       R5 = UCB address
                     28A1  8046 ;       R8 = Cancel reason code (CAN$C_DASSGN or CAN$C_CANCEL)
                     28A1  8047 ;
                     28A1  8048 ;
                     28A1  8049 ;       IPL = FIPL
                     28A1  8050 ;
                     28A1  8051 ; Outputs:
                     28A1  8052 ;
                     28A1  8053 ;       R3-R5 are preserved.
                     28A1  8054 ;       R0-R2 are destroyed.
                     28A1  8055 ;
                     28A1  8056 ;--
                     28A1  8057
                     28A1  8058 CANCEL::                                     ; Cancel I/O
00D8 8F     BB       28A1  8059          PUSHR   #^M<R3,R4,R6,R7>           ; Save registers
       03   E1       28A5  8060          BBC     #UCB$V_XE_SHARE,-          ; Br if not a shared UCB
17 68 A5             28A7  8061                   UCB$W_DEVSTS(R5),2$       ;  perform regular $CANCEL
                     28AA  8062 ;
                     28AA  8063 ; Try to find SHR data structure
                     28AA  8064 ;
    012A     30      28AA  8065          BSBW    FIND_SHR                   ; Check PID and CHAN
      12     12      28AD  8066          BNEQ    2$                         ; Br if NO MATCH, maybe last $DASSGN
                     28AF  8067 ;
                     28AF  8068 ;       Match found - clear inited bit and clean up all I/O on SHR data
                     28AF  8069 ;       structure.
                     28AF  8070 ;
                     28AF  8071 ;       We will Delete the SHR structure if this is a $DASSGN function
                     28AF  8072 ;       request. We will get this function when called from SYS$DASSGN
                     28AF  8073 ;       system service and so we will have to delete the SHR structure
                     28AF  8074 ;       and decrement the reference count. Note that the reference count
                     28AF  8075 ;       can never reach zero. Therefore, SYS$DASSGN will decrement the
                     28AF  8076 ;       reference count on exit and we will be called again. This time
                     28AF  8077 ;       there will be no match on the PID/CHAN and so the UCB will be
                     28AF  8078 ;       cleaned up and deleted.
                     28AF  8079 ;
   56   51   D0      28AF  8080          MOVL    R1,R6                      ; Copy SHR address
54 0180 C5   D0      28B2  8081          MOVL    UCB$L_XE_CDB(R5),R4        ; Get CDB address
    FF31     30      28B7  8082          BSBW    CLEANUP_SHR                ; Cleanup the SHR data structure
                     28BA  8083
                     28BA  8084          ASSUME  CAN$C_DASSGN EQ 1
      58     D7      28BA  8085          DECL    R8                         ; Deassign request?
      34     12      28BC  8086          BNEQ    10$                        ; Br if no - all done
    FF78     30      28BE  8087          BSBW    DELETE_SHR                 ; Else, delete the SHR data stucture
                     28C1  8088                                             ; And NOW perform like a NON-SHARED
                     28C1  8089                                             ; unit.
```

```
                                28C1  8090 ;
                                28C1  8091 ; Non-shared unit - perform $CANCEL function.
                                28C1  8092 ;
              5C A5    B5       28C1  8093 2$:      TSTW    UCB$W_REFC(R5)          ; Last reference?
                 31    12       28C4  8094          BNEQ    20$                     ; Br if no - do selective cancel
        54  0180 C5    D0       28C6  8095 3$:      MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
                FC8B    30       28CB  8096          BSBW    SHUTDOWN                ; Shutdown entire unit
                   00   E0       28CE  8097          BBS     #CDB_STS_V_INITED,-     ; Br if DEUNA/DELUA is still inited
           0A 025A C4           28D0  8098                  CDB_B_STS(R4),5$        ;
         0263 C4    01    CE    28D4  8099          MNEGL   #1,CDB_G_PHA(R4)        ; Reset physical address
         0267 C4    01    AE    28D9  8100          MNEGW   #1,CDB_G_PHA+4(R4)      ;       ...
                                28DE  8101 ;
                                28DE  8102 ; When this is the last reference to the unit, reset the CPID of the UCB.
                                28DE  8103 ;
                   10    A8       28DE  8104 5$:      BISW    S^#UCB$M_ONLINE,-       ; Set the UNIT to ONLINE
                 64 A5           28E0  8105                  UCB$W_STS(R5)           ;
                00BC C5    D5    28E2  8106          TSTL    UCB$L_XE_CPID(R5)       ; Did we save the Creator PID?
                   0A   13       28E6  8107          BEQL    10$                     ; Br if not
        20 A5   00BC C5    D0    28E8  8108          MOVL    UCB$L_XE_CPID(R5),UCB$L_CPID(R5) ; Else, restore Creator PID
                00BC C5    D4    28EE  8109          CLRL    UCB$L_XE_CPID(R5)       ; Never again!!
                00D8 8F    BA    28F2  8110 10$:     POPR    #^M<R3,R4,R6,R7>        ; Restore registers
                   05   28F6  8111          RSB
                                28F7  8112 ;
                                28F7  8113 ; Abort all associated receive packets on UCB queue
                                28F7  8114 ;
              54   04 AE    D0       28F7  8115 20$:     MOVL    4(SP),R4                ; Restore PCB address to R4
                   04   E1       28FB  8116          BBC     #UCB$V_ONLINE,-         ; Br if not online
                 F2 64 A5           28FD  8117                  UCB$W_STS(R5),10$       ;
                                2900  8118
                                2900  8119          ASSUME  UCB$V_XE_INITED EQ 0
                EE 68 A5    E9       2900  8120          BLBC    UCB$W_DEVSTS(R5),1$     ; Br if not inited
        56   00A8 C5    9E       2904  8121          MOVAB   UCB$Q_XE_RCVREQ(R5),R6  ; Get address of receive queue
                   29   10       2909  8122          BSBB    CHECKER                 ; Check packets on queue
                                290B  8123 ;
                                290B  8124 ; Abort all xmit requests on CDB queue
                                290B  8125 ;
        57  0180 C5    D0       290B  8126          MOVL    UCB$L_XE_CDB(R5),R7     ; Get CDB address
                   00   E1       2910  8127          BBC     #CDB_STS_V_INITED,-     ; Br if not inited
          DC 025A C7           2912  8128                  CDB_B_STS(R7),10$       ;
                                2916  8129          DSBINT  UCB$B_DIPL(R5)          ; Sync access to CDB
          57   7C A7    9E       291D  8130          MOVAB   CDB_Q_QUEUES(R7),R7     ; Get start of queues
             58   01    9A       2921  8131          MOVZBL  S^#CDB_C_ABORTS,R8      ; Get number of queues we can abort on
             56   57    D0       2924  8132 30$:     MOVL    R7,R6                   ; Set address of next queue
                  48    10       2927  8133          BSBB    CXB_CHECKER             ; Check CXBs on this queue
             57   08    C0       2929  8134          ADDL    #8,R7                   ; Skip to next queue
               F5 58    F5       292C  8135          SOBGTR  R8,30$                  ; Loop thru queues
                                292F  8136          ENBINT                          ; Enable interrupts
                   BE   11       2932  8137          BRB     10$                     ; Exit from cancel
                                2934  8138 ;
                                2934  8139 ; Subroutine to scan queue for match on all packets
                                2934  8140 ;
                                2934  8141 CHECKER:
             53   66    D0       2934  8142          MOVL    (R6),R3                 ; Get next entry
             56   53    D1       2937  8143 10$:     CMPL    R3,R6                   ; End of list?
                  11    13       293A  8144          BEQL    30$                     ; Br if yes
                  10    10       293C  8145          BSBB    CHECKPKT                ; Cancel if appropriate match
                  08    12       293E  8146          BNEQ    20$                     ; Br if no match
```

```
                53   63    0F   2940  8147          REMQUE  (R3),R3                 ; Remove from list
                   F5EE    30   2943  8148          BSBW    ABORT_PKT               ; Complete the I/O request
                     EC    11   2946  8149          BRB     CHECKER                 ; Look for more
                53   63    D0   2948  8150 20$:     MOVL    (R3),R3                 ; Travel link
                     EA    11   294B  8151          BRB     10$                     ; Look for more
                           05   294D  8152 30$:     RSB                             ; Return to caller
                                294E  8153 ;
                                294E  8154 ; Subroutine to check for specific cancel
                                294E  8155 ;
                                294E  8156 CHECKPKT:
              0C A3    D5   294E  8157          TSTL    IRP$L_PID(R3)           ; Is this an Internal IRP?
                 11    19   2951  8158          BLSS    30$                     ; Br if yes
                 06    12   2953  8159          BNEQ    10$                     ; Br if valid PID
                           2955  8160
                 54    D5   2955  8161          TSTL    R4                      ; Valid PCB?
                 17    12   2957  8162          BNEQ    50$                     ; Br if yes, no match
                 11    11   2959  8163          BRB     40$                     ; Else, test CHAN
                           295B  8164
           0C A3   60 A4    D1   295B  8165 10$:     CMPL    PCB$L_PID(R4),IRP$L_PID(R3) ; PID match?
                 0E    12   2960  8166          BNEQ    50$                     ; Br if no
                 08    11   2962  8167          BRB     40$                     ; Try CHAN match
                           2964  8168
         00B8 C5   60 A4    D1   2964  8169 30$:     CMPL    PCB$L_PID(R4),UCB$L_XE_PID(R5) ; IS this the starter's PID?
                 04    12   296A  8170          BNEQ    50$                     ; Br if no
              28 A3   52    B1   296C  8171 40$:     CMPW    R2,IRP$W_CHAN(R3)       ; Channel match?
                           05   2970  8172 50$:     RSB                             ; Return to caller
                                2971  8173
                                2971  8174 CXB_CHECKER:
           53   66    D0   2971  8175          MOVL    (R6),R3                 ; Get next entry
           56   53    D1   2974  8176 10$:     CMPL    R3,R6                   ; End of list?
                 2C    13   2977  8177          BEQL    30$                     ; Br if yes
                 2B    10   2979  8178          BSBB    CXB_CHECKPKT            ; Cancel if appropriate match
                 23    12   297B  8179          BNEQ    20$                     ; Br if no match
           53   63    0F   297D  8180          REMQUE  (R3),R3                 ; Remove from CXB list
        50   24 A3    D0   2980  8181          MOVL    CXB$L_T_IRP(R3),R0      ; Get IRP address
                 08    13   2984  8182          BEQL    16$                     ; Br if not, FAST interface
           53   50    D0   2986  8183          MOVL    R0,R3                   ; Copy IRP address
                F5A8    30   2989  8184          BSBW    ABORT_PKT               ; Complete the I/O request
                 E3    11   298C  8185          BRB     CXB_CHECKER             ; Look for more
                 54    DD   298E  8186 16$:     PUSHL   R4                      ; Else, save R4
     54   0178 C5    D0   2990  8187          MOVL    UCB$L_XE_FFI(R5),R4     ; Get FFI block address
           50   2C    3C   2995  8188          MOVZWL  #SS$_ABORT,R0           ; Set status return
              14 B4    16   2998  8189          JSB     @FFI$L_XMIT_DONE(R4)    ; Complete the XMIT CXB
                 54 8ED0   299B  8190          POPL    R4                      ; Restore R4
                 D1    11   299E  8191          BRB     CXB_CHECKER             ; Look for more
           53   63    D0   29A0  8192 20$:     MOVL    (R3),R3                 ; Travel link
                 CF    11   29A3  8193          BRB     10$                     ; Look for more
                           05   29A5  8194 30$:     RSB                             ; Return to caller
                                29A6  8195 ;
                                29A6  8196 ; Subroutine to check for specific cancel
                                29A6  8197 ;
                                29A6  8198 CXB_CHECKPKT:
        50   24 A3    D0   29A6  8199          MOVL    CXB$L_T_IRP(R3),R0      ; Get IRP address
                 23    13   29AA  8200          BEQL    80$                     ; Br if no IRP
              0C A0    D5   29AC  8201          TSTL    IRP$L_PID(R0)           ; Is this an Internal IRP?
                 11    19   29AF  8202          BLSS    30$                     ; Br if yes
                 06    12   29B1  8203          BNEQ    10$                     ; Br if valid PID
```

```
                        29B3  8204
                54   D5  29B3  8205          TSTL    R4                          ; Valid PCB?
                17   12  29B5  8206          BNEQ    50$                         ; Br if yes, no match
                11   11  29B7  8207          BRB     40$                         ; Else, test CHAN
                        29B9  8208
          0C A0   60 A4   D1  29B9  8209 10$:  CMPL    PCB$L_PID(R4),IRP$L_PID(R0) ; PID match?
                0E   12  29BE  8210          BNEQ    50$                         ; Br if no
                08   11  29C0  8211          BRB     40$                         ; Try CHAN match
                        29C2  8212
        00B8 C5   60 A4   D1  29C2  8213 30$:  CMPL    PCB$L_PID(R4),UCB$L_XE_PID(R5) ; IS this the starter's PID?
                04   12  29C8  8214          BNEQ    50$                         ; Br if no
           28 A0   52   B1  29CA  8215 40$:  CMPW    R2,IRP$W_CHAN(R0)           ; Channel match?
                05   29CE  8216 50$:  RSB                                       ; Return to caller
                        29CF  8217
                        29CF  8218          ; No IRP with CXB - FFI user
                        29CF  8219
                54   D5  29CF  8220 80$:  TSTL    R4                          ; No PCB?
                FB   13  29D1  8221          BEQL    50$                         ; Br if true - abort I/O
             53 53   D0  29D3  8222          MOVL    R3,R3                       ; Else, return Z-BIT clear
                05   29D6  8223          RSB                                 ; Return to caller
```

```
                        29D7  8225          .SBTTL   SUBROUTINES TO FIND SHR DATA STRUCTURE GIVEN PCB AND CHAN
                        29D7  8226 ;+
                        29D7  8227 ; Subroutine to find SHR data structure for user
                        29D7  8228 ;
                        29D7  8229 ; Inputs:
                        29D7  8230 ;       R2 = Channel number
                        29D7  8231 ;       R4 = PCB address (or zero)
                        29D7  8232 ;       R5 = UCB address
                        29D7  8233 ;
                        29D7  8234 ; Outputs:
                        29D7  8235 ;       R1 = Address if SHR data structure if match
                        29D7  8236 ;       R0 is destroyed.
                        29D7  8237 ;       Z-Bit set then match.
                        29D7  8238 ;       Z-Bit clear then no match.
                        29D7  8239 ;-
                        29D7  8240
                        29D7  8241 FIND_SHR:                                    ; Try to find shared user
    51   00C4 C5   D0   29D7  8242          MOVL    UCB$L_XE_DEFUSR(R5),R1      ; Get address of default user
               04   13   29DC  8243          BEQL    10$                         ; Br if no default user
               1C   10   29DE  8244          BSBB    90$                         ; Check for match
               19   13   29E0  8245          BEQL    40$                         ; Br if match
    50   0098 C5   9E   29E2  8246 10$:     MOVAB   UCB$Q_XE_SHARE(R5),R0       ; Save address of listhead
         51   50   D0   29E7  8247          MOVL    R0,R1                       ; Copy listhead address
                        29EA  8248          ASSUME  SHR_L_QFL EQ 0
         51   61   D0   29EA  8249 20$:     MOVL    (R1),R1                     ; Get next in list
         50   51   D1   29ED  8250          CMPL    R1,R0                       ; Back to start of list?
               06   13   29F0  8251          BEQL    30$                         ; Br if yes - no pid/chan match
               08   10   29F2  8252          BSBB    90$                         ; Check for match
               F4   12   29F4  8253          BNEQ    20$                         ; Br if none
               03   11   29F6  8254          BRB     40$                         ; Return in success
         50   50   D0   29F8  8255 30$:     MOVL    R0,R0                       ; Return match failure
               05   29FB  8256 40$:     RSB
                        29FC  8257
                        29FC  8258 ;+
                        29FC  8259 ; Subroutine to check if PID and SHR data base match up
                        29FC  8260 ;
                        29FC  8261 ; Inputs:
                        29FC  8262 ;       R1 = SHR address
                        29FC  8263 ;       R2 = Channel number
                        29FC  8264 ;       R4 = PCB address (or zero)
                        29FC  8265 ;
                        29FC  8266 ; Outputs:
                        29FC  8267 ;       Z-Bit set then match.
                        29FC  8268 ;       Z-Bit clear then no match.
                        29FC  8269 ;-
                        29FC  8270
                        29FC  8271 90$:                                         ; Check for match with SHR data base
               54   D5   29FC  8272          TSTL    R4                          ; Valid PCB address?
               07   12   29FE  8273          BNEQ    100$                        ; Br if yes
            0C A1   D5   2A00  8274          TSTL    SHR_L_PID(R1)               ; Zero PID?
               0D   12   2A03  8275          BNEQ    140$                        ; Br if not
               07   11   2A05  8276          BRB     110$                        ; Try for CHAN
 0C A1     60 A4   D1   2A07  8277 100$:    CMPL    PCB$L_PID(R4),SHR_L_PID(R1) ; PIDs match?
               04   12   2A0C  8278          BNEQ    140$                        ; Br if no - try for next
    10 A1      52   B1   2A0E  8279 110$:    CMPW    R2,SHR_W_CHAN(R1)           ; Channels match?
               05   2A12  8280 140$:    RSB                                     ; Return to caller
```

```
                        2A13  8282          .SBTTL  FIND_POINT_UCB  - Find the point to point UCB
                        2A13  8283 ;++
                        2A13  8284 ; FIND_POINT_UCB - Find the point-to-point UCB
                        2A13  8285 ;
                        2A13  8286 ; Functional description:
                        2A13  8287 ;
                        2A13  8288 ; This routine is called to find the point-to-point UCB for some received
                        2A13  8289 ; message. This is only needed when the protocol is in the startup state.
                        2A13  8290 ;
                        2A13  8291 ; Inputs:
                        2A13  8292 ;
                        2A13  8293 ;       R1 = Protocol type (startup)
                        2A13  8294 ;       R2 = MSG buffer address
                        2A13  8295 ;       R4 = CDB address
                        2A13  8296 ;
                        2A13  8297 ;
                        2A13  8298 ;       IPL = FIPL
                        2A13  8299 ;
                        2A13  8300 ; Outputs:
                        2A13  8301 ;
                        2A13  8302 ;       R0 = Status return for request
                        2A13  8303 ;       All other registers are preserved
                        2A13  8304 ;--
                        2A13  8305
                        2A13  8306 FIND_POINT_UCB:                                      ;% Find the point-to-point UCB
     006E 8F    BB      2A13  8307          PUSHR   #^M<R1,R2,R3,R5,R6>                 ;% Save registers
          50    D4      2A17  8308          CLRL    R0                                  ;% Assume failure
   46 A2    01  B1      2A19  8309          CMPW    #1,CXB$W_R_SIZE(R2)                 ;% Is buffer the right size?
          54    12      2A1D  8310          BNEQ    99$                                 ;% Br if not!
48 A2    AA 8F  91      2A1F  8311          CMPB    #^XAA,CXB$W_R_SIZE+2(R2)            ;% Is first byte start byte?
          07    13      2A24  8312          BEQL    10$                                 ;% Br if yes
48 A2    AB 8F  91      2A26  8313          CMPB    #^XAB,CXB$W_R_SIZE+2(R2)            ;% Is first byte stack byte?
          73    12      2A2B  8314          BNEQ    90$                                 ;% Br if not
 55   00C8 C4    D0      2A2D  8315 10$:     MOVL    CDB_L_UCB0(R4),R5                   ;% Get UCB address of unit 0
   55   30 A5    D0      2A32  8316 20$:     MOVL    UCB$L_LINK(R5),R5                   ;% Get address of next UCB
          68    13      2A36  8317          BEQL    90$                                 ;% Br if end of list
                        2A38  8318          ASSUME  UCB$V_XE_INITED EQ 0
     F6 68 A5    E9      2A38  8319          BLBC    UCB$W_DEVSTS(R5),20$                ;% Br if not inited
          00    91      2A3C  8320          CMPB    #NMA$C_LINPR_POI,-                  ;% Is this a point-to-point user?
     00D8 C5              2A3E  8321                  UCB$B_XE_PRO(R5)                    ;%
          EF    12      2A41  8322          BNEQ    20$                                 ;% Br if not
     00CC C5    D1      2A43  8323          CMPL    UCB$G_XE_DES(R5),-                  ;% Does the destination match?
       3E A2              2A47  8324                  CXB$G_R_SRC(R2)                     ;%
          E7    12      2A49  8325          BNEQ    20$                                 ;% Br if not
     00D0 C5    B1      2A4B  8326          CMPW    UCB$G_XE_DES+4(R5),-                ;% Still match?
       42 A2              2A4F  8327                  CXB$G_R_SRC+4(R2)                   ;%
          DF    12      2A51  8328          BNEQ    20$                                 ;% Br if not
48 A2    AB 8F  91      2A53  8329          CMPB    #^XAB,CXB$W_R_SIZE+2(R2)            ;% Is first byte stack byte?
          14    12      2A58  8330          BNEQ    40$                                 ;% Br if no
05 68 A5    05  E4      2A5A  8331          BBSC    #UCB$V_XE_START,UCB$W_DEVSTS(R5),30$ ;% Clear Start state
28 68 A5    06  E5      2A5F  8332          BBCC    #UCB$V_XE_STACK,UCB$W_DEVSTS(R5),60$ ;% We were in RUN, ignore
 52   017C C5    D0      2A64  8333 30$:     MOVL    UCB$L_XE_STIRP(R5),R2               ;% Get start IRP
        FD0E    30      2A69  8334          BSBW    BLD_STRT_IRP                        ;% Send stack!
          1E    11      2A6C  8335          BRB     60$                                 ;% Then send data
48 A2    AA 8F  91      2A6E  8336 40$:     CMPB    #^XAA,CXB$W_R_SIZE+2(R2)            ;% Is first byte start byte?
          2B    12      2A73  8337 99$:     BNEQ    90$                                 ;% Br if not, ignore it
     0060 8F    B3      2A75  8338          BITW    #UCB$M_XE_START!UCB$M_XE_STACK,-    ;% Are we in startup states?
```

```
                68 A5        2A79  8339                    UCB$W_DEVSTS(R5)
                   28  13    2A7B  8340            BEQL    120$                            ;% Br if not, start recvd in RUN
                           2A7D  8341            CLRBIT  #UCB$V_XE_START,UCB$W_DEVSTS(R5);% Clear starting bit
           52   017C C5     D0  2A82  8342            MOVL    UCB$L_XE_STIRP(R5),R2           ;% Get start IRP
                  FCF0  30     2A87  8343            BSBW    BLD_STRT_IRP                    ;% Send stack!
                     11  11    2A8A  8344            BRB     80$                             ;% Wait for stack
           53   00B0 D5     0F  2A8C  8345 60$:      REMQUE  @UCB$Q_XE_XMTREQ(R5),R3         ;% Get transmit IRPs
                     07  1D    2A91  8346            BVS     70$                             ;% Br if no more
           0080 D4     63  0E  2A93  8347            INSQUE  (R3),@CDB_Q_XMTREQ+4(R4)        ;% Insert IRPs onto xmit queue
                     F2  11    2A98  8348            BRB     60$                             ;% Look for more
                  DA0B   30     2A9A  8349 70$:      BSBW    XMT_ALT_START                   ;% Startup the xmit process
                50   01  9A    2A9D  8350 80$:      MOVZBL  #1,R0                           ;% Return success
                           2AA0  8351
                006E 8F     BA  2AA0  8352 90$:      POPR    #^M<R1,R2,R3,R5,R6>             ;% Restore registers
                     05  2AA4  8353            RSB                                     ;% Return to caller
                           2AA5  8354
                           2AA5  8355 ;
                           2AA5  8356 ; Start received in run mode
                           2AA5  8357 ;
                0098 C5     D1  2AA5  8358 120$:     CMPL    UCB$Q_XE_SHARE(R5),-            ;% Is limited queue empty?
                0098 D5         2AA9  8359                    @UCB$Q_XE_SHARE(R5)             ;%
                     F2  13    2AAC  8360            BEQL    90$                             ;% Br if yes, no IRPs
                           2AAE  8361            CLRBIT  #XM$V_STS_ACTIVE,UCB$L_DEVDEPEND(R5) ;% Clear active bit
                           2AB3  8362            SETBIT  #XM$V_ERR_START,UCB$L_DEVDEPEND(R5) ;% Indicate cause of error
                0060 8F     AA  2AB8  8363            BICW    #UCB$M_XE_START!UCB$M_XE_STACK,-;% Clear start and stack flags
                  68 A5         2ABC  8364                    UCB$W_DEVSTS(R5)                ;%
                           2ABE  8365            CLRBIT  #UCB$V_XE_RUN,UCB$W_DEVSTS(R5)  ;% Clear the RUN flag
           56   0098 C5     D0  2AC3  8366            MOVL    UCB$Q_XE_SHARE(R5),R6           ;% Get address of share structure
                  FD20  30     2AC8  8367            BSBW    CLEANUP_SHR                     ;% Cleanup all pending I/O
                     D3  11    2ACB  8368            BRB     90$                             ;% Exit
                           2ACD  8369
```

```
                     2ACD  8371          .SBTTL  ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
                     2ACD  8372 ;++
                     2ACD  8373 ; ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
                     2ACD  8374 ;
                     2ACD  8375 ; Functional description:
                     2ACD  8376 ;
                     2ACD  8377 ; This routine is called to combine all the per protocol type multicast
                     2ACD  8378 ; addresses into a single list in the CDB. If the sum of all multicast
                     2ACD  8379 ; addresses is greater than the DEUNA/DELUA can manage, then an error is
                     2ACD  8380 ; returned.
                     2ACD  8381 ;
                     2ACD  8382 ; Inputs:
                     2ACD  8383 ;
                     2ACD  8384 ;       R5 = UCB address
                     2ACD  8385 ;
                     2ACD  8386 ;
                     2ACD  8387 ;       IPL = FIPL
                     2ACD  8388 ;
                     2ACD  8389 ; Outputs:
                     2ACD  8390 ;
                     2ACD  8391 ;       R0 = Status return for request
                     2ACD  8392 ;       R1,R2 are destroyed
                     2ACD  8393 ;       R3-R5 are preserved
                     2ACD  8394 ;
                     2ACD  8395 ; Implicit outputs:
                     2ACD  8396 ;
                     2ACD  8397 ;       CDB_B_MLTTBL = Number of multicast addresses in CDB_G_MLTTBL
                     2ACD  8398 ;       CDB_G_MLTTBL = New multicast address list
                     2ACD  8399 ;--
                     2ACD  8400
                     2ACD  8401 ADD_MULTI:                              ; Add up all the multicast addresses
          00D8 8F    BB   2ACD  8402          PUSHR   #^M<R3,R4,R6,R7>   ; Save registers
     54   0180 C5    D0   2AD1  8403          MOVL    UCB$L_XE_CDB(R5),R4 ; Get CDB address
          0277 C4    94   2AD6  8404          CLRB    CDB_B_MLTTBL(R4)   ; Reset number of entries
                     2ADA  8405          PUSHQ   R4                 ; Save CDB and UCB addresses
00   0278 C4    00   2C   2ADD  8406          MOVC5   #0,CDB_G_MLTTBL(R4),#0.- ; Zero the structure
     0278 C4    3C        2AE3  8407                  #6*MAX_C_MLT,CDB_G_MLTTBL(R4) ;
                     2AE7  8408          POPQ    R4                 ; Restore CDB and UCB addresses
          53    0B   9A   2AEA  8409          MOVZBL  #MAX_C_MLT+1,R3    ; Error if 1 more multicast address
                     2AED  8410                                     ;  than we can handle
          50    01   CE   2AED  8411          MNEGL   #1,R0              ; Assume success
     56   0278 C4    9E   2AF0  8412          MOVAB   CDB_G_MLTTBL(R4),R6 ; Get address of Multicast table
       57   28 A5    D0   2AF5  8413          MOVL    UCB$L_DDB(R5),R7   ; Get DDB address
       57   04 A7    D0   2AF9  8414          MOVL    DDB$L_UCB(R7),R7   ; Get 1st UCB address
       57   30 A7    D0   2AFD  8415 10$:     MOVL    UCB$L_LINK(R7),R7  ; Get next UCB in list
                37    13   2B01  8416          BEQL    50$                ; Br if no more UCB's
                     2B03  8417
                     2B03  8418          ASSUME  UCB$V_XE_INITED  EQ  0
          F6 68 A5   E9   2B03  8419          BLBC    UCB$W_DEVSTS(R5),10$ ; Br if not inited
     52   00E9 C7    9E   2B07  8420          MOVAB   UCB$G_XE_MULTI(R7),R2 ; Get address of Multicast list
     51   00E7 C7    9A   2B0C  8421          MOVZBL  UCB$B_XE_MULTI(R7),R1 ; Set number addresses for UCB
                62    D5   2B11  8422 20$:     TSTL    (R2)               ; Is this field unused?
                05    12   2B13  8423          BNEQ    25$                ; Br if no
             04 A2    B5   2B15  8424          TSTW    4(R2)              ; Really?
                16    13   2B18  8425          BEQL    30$                ; Yes - skip it
                53    97   2B1A  8426 25$:     DECB    R3                 ; One less available slot in CDB
                1A    13   2B1C  8427          BEQL    40$                ; Br if none left - error
```

```
                86   62    D0   2B1E  8428          MOVL    (R2),(R6)+              ; Else, insert next address
          86     04  A2    B0   2B21  8429          MOVW    4(R2),(R6)+             ;       ...
              0277  C4    96   2B25  8430          INCB    CDB_B_MLTTBL(R4)        ; Count one more in list
              0277  C4    91   2B29  8431          CMPB    CDB_B_MLTTBL(R4),-      ; Is there enough room?
                    0A          2B2D  8432           #MAX_C_MLT                     ;       ...
                    08    1A    2B2E  8433          BGTRU   40$                     ; Br if no - error
                52  06    C0   2B30  8434  30$:     ADDL    #6,R2                   ; Skip to next entry
                  DB  51  F5   2B33  8435          SOBGTR  R1,20$                  ; Br if more
                      C5  11   2B36  8436          BRB     10$                     ; Else, skip to next UCB
                          2B38  8437
                      50  D4   2B38  8438  40$:     CLRL    R0                      ; Return failure
              00D8    8F  BA   2B3A  8439  50$:     POPR    #^M<R3,R4,R6,R7>        ; Restore registers
                          05   2B3E  8440          RSB
```

```
                                2B3F  8442          .SBTTL  MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
                                2B3F  8443 ;++
                                2B3F  8444 ; MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
                                2B3F  8445 ;
                                2B3F  8446 ; Functional description:
                                2B3F  8447 ;
                                2B3F  8448 ; This routine is called to copy the multicast address list from the
                                2B3F  8449 ; generation table to the actual list.
                                2B3F  8450 ;
                                2B3F  8451 ; Inputs:
                                2B3F  8452 ;
                                2B3F  8453 ;       R4 = CDB address
                                2B3F  8454 ;
                                2B3F  8455 ; Outputs:
                                2B3F  8456 ;
                                2B3F  8457 ;       All registers are preserved.
                                2B3F  8458 ;
                                2B3F  8459 ; Implicit outputs:
                                2B3F  8460 ;
                                2B3F  8461 ;       CDB_B_MULTI = Number of multicast addresses in CDB_G_MUDB
                                2B3F  8462 ;       CDB_G_MUDB = New multicast address list
                                2B3F  8463 ;--
                                2B3F  8464
                                2B3F  8465 MOVE_MULTI:                          ; Move the multicast address list
                          3F    BB  2B3F  8466          PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save registers
012A C4   0278 C4         3C    28  2B41  8467          MOVC3   #MAX_C_MLT*6,CDB_G_MLTTBL(R4),CDB_G_MUDB(R4) ; Copy list
                          3F    BA  2B49  8468          POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore registers
     0276 C4   0277 C4    90        2B4B  8469          MOVB    CDB_B_MLTTBL(R4),CDB_B_MULTI(R4) ; Set number of entries
                                05  2B52  8470          RSB
```

```
                     2B53  8472          .SBTTL  ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
                     2B53  8473 ;++
                     2B53  8474 ; ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
                     2B53  8475 ;
                     2B53  8476 ; Functional description:
                     2B53  8477 ;
                     2B53  8478 ; These routines are called to save or restore the multicast address list
                     2B53  8479 ; in the UCB.
                     2B53  8480 ;
                     2B53  8481 ; Inputs:
                     2B53  8482 ;
                     2B53  8483 ;       R5 = UCB address
                     2B53  8484 ;
                     2B53  8485 ; Outputs:
                     2B53  8486 ;
                     2B53  8487 ;       All registers are preserved.
                     2B53  8488 ;--
                     2B53  8489
                     2B53  8490 SAV_MULTI:                                      ; Save multicast address list
          3F   BB    2B53  8491          PUSHR   #^M<R0,R1,R2,R3,R4,R5>        ; Save registers
00E9 C5    3C   28    2B55  8492          MOVC3   #6*MAX_C_MLT,UCB$G_XE_MULTI(R5),- ; Save multicast addresses
     0125 C5          2B5A  8493                  UCB$G_XE_MLTTBL(R5)           ;       ...
          3F   BA    2B5D  8494          POPR    #^M<R0,R1,R2,R3,R4,R5>        ; Restore registers
     00E7 C5   90    2B5F  8495          MOVB    UCB$B_XE_MULTI(R5),-          ; Save count of multicast addresses
     00E8 C5         2B63  8496                  UCB$B_XE_MLTTBL(R5)           ;       ...
               05    2B66  8497          RSB                                   ; Return to caller
                     2B67  8498
                     2B67  8499 RES_MULTI:                                      ; Restore multicast address list
          3F   BB    2B67  8500          PUSHR   #^M<R0,R1,R2,R3,R4,R5>        ; Save registers
0125 C5    3C   28    2B69  8501          MOVC3   #6*MAX_C_MLT,UCB$G_XE_MLTTBL(R5),- ; Restore multicast addresses
     00E9 C5         2B6E  8502                  UCB$G_XE_MULTI(R5)            ;       ...
          3F   BA    2B71  8503          POPR    #^M<R0,R1,R2,R3,R4,R5>        ; Restore registers
     00E8 C5   90    2B73  8504          MOVB    UCB$B_XE_MLTTBL(R5),-         ; Restore count of multicast addresses
     00E7 C5         2B77  8505                  UCB$B_XE_MULTI(R5)            ;       ...
               05    2B7A  8506          RSB                                   ; Return to caller
```

```
                     2B7B  8508          .SBTTL  VALIDATE_P2 - VALIDATE P2 BUFFER PARAMETERS
                     2B7B  8509
                     2B7B  8510 ;++
                     2B7B  8511 ; VALIDATE_P2 - Validate P2 buffer parameters
                     2B7B  8512 ;
                     2B7B  8513 ; This routine is called to validate the P2 buffer parameters. The parameters
                     2B7B  8514 ; are checked against a parameter table which verifies that the minimum value
                     2B7B  8515 ; and maximum value is not violated, and that invalid status flags are not set.
                     2B7B  8516 ; If the parameter is a string, then the string must not exceed the maximum
                     2B7B  8517 ; string count for this parameter.
                     2B7B  8518 ;
                     2B7B  8519 ; Inputs:
                     2B7B  8520 ;
                     2B7B  8521 ;       R2 = Address of verification table
                     2B7B  8522 ;       R3 = IRP address
                     2B7B  8523 ;       R5 = UCB address
                     2B7B  8524 ;
                     2B7B  8525 ;
                     2B7B  8526 ; Outputs:
                     2B7B  8527 ;
                     2B7B  8528 ;       R0 = Status return for request
                     2B7B  8529 ;
                     2B7B  8530 ; If no error:
                     2B7B  8531 ;       R1 = Address of parameter verification table
                     2B7B  8532 ; If error:
                     2B7B  8533 ;       R1 = Bad parameter value
                     2B7B  8534 ;
                     2B7B  8535 ;       All other registers are preserved.
                     2B7B  8536 ;
                     2B7B  8537 ;--
                     2B7B  8538
                     2B7B  8539 VALIDATE_P2::                           ; Validate P2 buffer parameters
        03DC 8F   BB 2B7B  8540         PUSHR   #^M<R2,R3,R4,R6,R7,R8,R9> ; Save registers
   56     2C A3   D0 2B7F  8541         MOVL    IRP$L_SVAPTE(R3),R6       ; Get system P2 buffer address
          03   12 2B83  8542         BNEQ    20$                       ; Br if a system buffer
        0152   31 2B85  8543 10$:      BRW     150$                      ; Else, leave
                     2B88  8544
   54   0180 C5   D0 2B88  8545 20$:      MOVL    UCB$L_XE_CDB(R5),R4       ; Get CDB address
        56   66   D0 2B8D  8546         MOVL    P2B_L_POINTER(R6),R6      ; Point to start of P2 data
   58     32 A3   3C 2B90  8547         MOVZWL  IRP$W_BCNT(R3),R8         ; Get size of P2 buffer
                     2B94  8548 ;
                     2B94  8549 ; Loop to check next parameter in P2 buffer
                     2B94  8550 ;
        58   02   C2 2B94  8551 30$:      SUBL    #2,R8                     ; Can we get another parameter?
             EC   19 2B97  8552         BLSS    10$                       ; Br if no - all done
        51   86   3C 2B99  8553         MOVZWL  (R6)+,R1                  ; Get parameter type from P2
                     2B9C  8554 ;
                     2B9C  8555 ; *** NOTE - R2 MUST be the very first item on the top of the stack
                     2B9C  8556 ;
        57   6E   D0 2B9C  8557         MOVL    (SP),R7                   ; Get verification table address
                     2B9F  8558 ;
                     2B9F  8559 ; Loop to check P2 buffer parameter to Line parameter table
                     2B9F  8560 ;
                     2B9F  8561         ASSUME  PRM_W_TYPE EQ 0
        50   87   B0 2B9F  8562 40$:      MOVW    (R7)+,R0                  ; Get parameter type code
             03   12 2BA2  8563         BNEQ    45$                       ; Br if NOT end of verify table
          0138   31 2BA4  8564 43$:      BRW     170$                      ; Else exit in error
```

```
                        2BA7  8565
                        2BA7  8566 45$:     ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
      59   87   9A       2BA7  8567          MOVZBL  (R7)+,R9                 ; Get flags byte
50    F000 8F   AA       2BAA  8568          BICW    #^C<PRM_TYP_M_CODE>,R0  ; Clear all but type code
      50   51   B1       2BAF  8569          CMPW    R1,R0                   ; Parameters match?
           17   13       2BB2  8570          BEQL    50$                     ; Br if yes
      57   02   C0       2BB4  8571          ADDL    #2,R7                   ; Skip offset word
                        2BB7  8572          SKIP    PRM_FLG_V_MIN,R9,R7     ; Skip minimum value
                        2BBD  8573          SKIP    PRM_FLG_V_MAX,R9,R7     ; Skip maximum value
                        2BC3  8574          SKIP    PRM_FLG_V_INVALID,R9,R7 ; Skip invalid flags
           D4   11      2BC9  8575          BRB     40$                     ; Try next parameter
                        2BCB  8576 ;
                        2BCB  8577 ; Match found - nullify if same value & check min,max,valid,invalid
                        2BCB  8578 ;
      50   87   B0       2BCB  8579 50$:     MOVW    (R7)+,R0                ; Get offset + width
           0A   EF       2BCE  8580          EXTZV   #PRM_OFF_V_WIDTH,-      ; Get width only
52    50   06             2BD0  8581                  #PRM_OFF_S_WIDTH,R0,R2  ;   ...
           00   EF       2BD3  8582          EXTZV   #PRM_OFF_V_VALUE,-      ; Get offset only
50    50   0A             2BD5  8583                  #PRM_OFF_S_VALUE,R0,R0  ;   ...
  05  59   03   E0       2BD8  8584          BBS     #PRM_FLG_V_CDB,R9,55$   ; Br if CDB datum
      50   55   C0       2BDC  8585          ADDL    R5,R0                   ; Compute offset in UCB
           03   11       2BDF  8586          BRB     57$                     ; Continue
      50   54   C0       2BE1  8587 55$:     ADDL    R4,R0                   ; Compute offset in CDB
                        2BE4  8588 57$:     ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
31 FB A7   0C   E0       2BE4  8589          BBS     #PRM_TYP_V_STRING,-5(R7),95$ ; Br if string parameter
      58   04   C2       2BE9  8590          SUBL    #4,R8                   ; Must be longword value
           B6   19       2BEC  8591          BLSS    43$                     ; Br if error
      53   86   D0       2BEE  8592          MOVL    (R6)+,R3                ; Get parameter value
                        2BF1  8593          CASE    R2,TYPE=B,LIMIT=#1,<-   ; Br to handler
                        2BF1  8594                  60$,-                   ;   Byte value
                        2BF1  8595                  70$,-                   ;   Word value
                        2BF1  8596                  80$>                    ;   Longword value
                        2BFB  8597 ;
                        2BFB  8598 ; Byte value in structure
                        2BFB  8599 ;
      60   53   91       2BFB  8600 60$:     CMPB    R3,(R0)                 ; Is this the same?
           08   11       2BFE  8601          BRB     90$                     ; Check result
                        2C00  8602 ;
                        2C00  8603 ; Word value
                        2C00  8604 ;
      60   53   B1       2C00  8605 70$:     CMPW    R3,(R0)                 ; Is this the same?
           03   11       2C03  8606          BRB     90$                     ; Check result
                        2C05  8607 ;
                        2C05  8608 ; Longword value
                        2C05  8609 ;
      60   53   D1       2C05  8610 80$:     CMPL    R3,(R0)                 ; Is this the same?
           6F   12       2C08  8611 90$:     BNEQ    100$                    ; Br if no - continue checks
0B0E  8F   51   B1       2C0A  8612          CMPW    R1,#NMA$C_PCLI_PTY      ; Is this the protocol type?
           03   13       2C0F  8613          BEQL    91$                     ; Br if yes - always store this
        FA A6   B4       2C11  8614          CLRW    -6(R6)                  ; Nullify the parameter code
         00C0   31       2C14  8615 91$:     BRW     145$                    ; Try next parameter - skip checks
         00C5   31       2C17  8616 93$:     BRW     170$                    ; LONNGGG Branch to 170$
                        2C1A  8617 ;
                        2C1A  8618 ; String value
                        2C1A  8619 ;
      58   02   C2       2C1A  8620 95$:     SUBL    #2,R8                   ; Can we fetch string length?
           F8   19       2C1D  8621          BLSS    93$                     ; Br if no - error
```

```
                53   86   3C   2C1F   8622          MOVZWL  (R6)+,R3                   ; Get string length
                58   53   C2   2C22   8623          SUBL    R3,R8                     ; Is there room for string?
                     F0   19   2C25   8624          BLSS    93$                       ; Br if no - error
                52   53   B1   2C27   8625          CMPW    R3,R2                     ; Is the string too long?
                     EB   1A   2C2A   8626          BGTRU   93$                       ; Br if yes - error
                56   53   C0   2C2C   8627          ADDL    R3,R6                     ; Skip past string
           51   0B21 8F   B1   2C2F   8628          CMPW    #NMA$C_PCLI_DES,R1        ; Is this the destination address?
                     07   13   2C34   8629          BEQL    96$                       ; Br if yes
           51   0B04 8F   B1   2C36   8630          CMPW    #NMA$C_PCLI_PHA,R1        ; Is this the physical address?
                     09   12   2C3B   8631          BNEQ    97$                       ; Br if not
                  029B   30   2C3D   8632 96$:      BSBW    VALID_PHYAD               ; Validate the physical address
                D4 50    E9   2C40   8633          BLBC    R0,93$                    ; Br if error in physical address
                  0091   31   2C43   8634          BRW     145$                      ; Else, continue checking
           51   0B0F 8F   B1   2C46   8635 97$:      CMPW    #NMA$C_PCLI_MCA,R1        ; Is this the multicast address list?
                     3E   12   2C4B   8636          BNEQ    130$                      ; Br if no - okay
                  024F   30   2C4D   8637          BSBW    VALID_MULTI               ; Validate the multicast address list
                C4 50    E9   2C50   8638          BLBC    R0,93$                    ; Br if error
                          2C53   8639          DSBINT  UCB$B_FIPL(R5)            ; Sync access to UCB
                  FEF6   30   2C5A   8640          BSBW    SAV_MUL,I                 ; Save the multicast addresses
                0240 8F   BB   2C5D   8641          PUSHR   #^M<R6,R9>                ; Save registers
                56   53   C2   2C61   8642          SUBL    R3,R6                     ; Backup pointer to start of list
                59   53   D0   2C64   8643          MOVL    R3,R9                     ; Setup string count in R9
                  02A9   30   2C67   8644          BSBW    SET_MULTI                 ; See if we can set new addresses
                          2C6A   8645                                                ; R0 = return status
                0240 8F   BA   2C6A   8646          POPR    #^M<R6,R9>                ; Restore registers
                  FEF6   30   2C6E   8647          BSBW    RES_MULTI                 ; Restore the multicast list
                          2C71   8648          ENBINT                            ; Restore IPL
                68 50    E9   2C74   8649          BLBC    R0,170$                   ; Br if error
                     12   11   2C77   8650          BRB     130$                      ; Check if state okay
                          2C79   8651
          05 59    00   E1   2C79   8652 100$:     BBC     #PRM_FLG_V_MIN,R9,110$    ; Br if no minimum value
                87   53   B1   2C7D   8653          CMPW    R3,(R7)+                  ; Is the value too small?
                     5D   1F   2C80   8654          BLSSU   170$                      ; Br if yes - error
          05 59    01   E1   2C82   8655 110$:     BBC     #PRM_FLG_V_MAX,R9,130$    ; Br if no maximum value
                87   53   B1   2C86   8656          CMPW    R3,(R7)+                  ; Is the value too big?
                     54   1A   2C89   8657          BGTRU   170$                      ; Br if yes - error
          48 59    02   E1   2C8B   8658 130$:     BBC     #PRM_FLG_V_INVALID,R9,145$ ; Br if no invalid flags
                52   87   B0   2C8F   8659          MOVW    (R7)+,R2                  ; Get invalid flags
          06 59    03   E0   2C92   8660          BBS     #PRM_FLG_V_CDB,R9,135$    ; Br if CDB datum
          68 A5    52   B3   2C96   8661          BITW    R2,UCB$W_DEVSTS(R5)       ; Check UCB invalid bits
                     09   11   2C9A   8662          BRB     137$                      ; Continue
                     54   D5   2C9C   8663 135$:     TSTL    R4                        ; Is CDB present?
                     07   13   2C9E   8664          BEQL    140$                      ; Br if no - okay
     025A C4         52   93   2CA0   8665          BITB    R2,CDB_B_STS(R4)          ; Check CDB invalid bits
                     38   12   2CA5   8666 137$:     BNE.    170$                      ; Br on error
                          2CA7   8667
          50    0A A5    9A   2CA7   8668 140$:     MOVZBL  UCB$B_TYPE(R5),R0         ; Retrieve device type
                          2CAB   8669          $DISPATCH       R0,TYPE=B,-       ; Dispatch based upon device type
                          2CAB   8670          < -
                          2CAB   8671                  <DT$_DEUNA      141$>,- ; DEUNA - don't allow NMA$C_PCLI_ILP
                          2CAB   8672                  <DT$_DELUA      143$>,- ; DELUA - don't allow NMA$C_PCLI_EKO
                          2CAB   8673          >
                          2CC7   8674
     0B17 8F         51   B1   2CC7   8675 141$:     CMPW    R1,#NMA$C_PCLI_ILP        ; Is this the ILP Line Parameter
                     11   13   2CCC   8676          BEQL    170$                      ; Return bad parameter if so
                     07   11   2CCE   8677          BRB     145$                      ; Else see if there are more parameters
                          2CD0   8678
```

```
                0B1F 8F    51    B1  2CD0  8679 143$:   CMPW    R1,#NMA$C_PCLI_EKO      ; Is this the EKO Line Parameter
                           08    13  2CD5  8680         BEQL    170$                    ; Return bad parameter if so
                       FEBA      31  2CD7  8681 145$:   BRW     30$                     ; Loop if more parameters
                                     2CDA  8682
                     50    01    9A  2CDA  8683 150$:   MOVZBL  S^#SS$_NORMAL,R0        ; Set success return
                           03    11  2CDD  8684         BRB     180$                    ; And return
                                     2CDF  8685
                     50    14    9A  2CDF  8686 170$:   MOVZBL  S^#SS$_BADPARAM,R0      ; Set error return
                     03DC  8F    BA  2CE2  8687 180$:   POPR    #^M<R2,R3,R4,R6,R7,R8,R9> ; Restore registers
                                 05  2CE6  8688         RSB                             ; Return to caller
```

```
                     2CE7  8690          .SBTTL  CHANGE_PARAM - UPDATE UCB/CDB BASED ON P2 BUFFER PARAMETERS
                     2CE7  8691
                     2CE7  8692 ;++
                     2CE7  8693 ; CHANGE_PARAM - Update UCB/CDB with P2 buffer parameters
                     2CE7  8694 ;
                     2CE7  8695 ; This routine is called to update the UCB/CDB with the P2 buffer parameters.
                     2CE7  8696 ; The parameters are stored in the appropriate cells of the UCB/CDB.
                     2CE7  8697 ; This routine can only modify the LINE PARAMETERS.
                     2CE7  8698 ;
                     2CE7  8699 ; Inputs:
                     2CE7  8700 ;
                     2CE7  8701 ;       R2 = Address of verification table
                     2CE7  8702 ;       R3 = IRP address
                     2CE7  8703 ;       R5 = UCB address
                     2CE7  8704 ;
                     2CE7  8705 ;       IPL = FIPL
                     2CE7  8706 ;
                     2CE7  8707 ; Outputs:
                     2CE7  8708 ;
                     2CE7  8709 ;       R0 = destroyed.
                     2CE7  8710 ;       All other registers are preserved.
                     2CE7  8711 ;
                     2CE7  8712 ;--
                     2CE7  8713
                     2CE7  8714 CHANGE_PARAM:                           ; Change the UCB/CDB parameters
     07DE 8F    BB   2CE7  8715          PUSHR   #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Save registers
     5A   52    D0   2CEB  8716          MOVL    R2,R10                  ; Save table address
  56   2C A3    D0   2CEE  8717          MOVL    IRP$L_SVAPTE(R3),R6     ; Get system P2 buffer address
        03    12   2CF2  8718          BNEQ    5$                      ; Br if system buffer
       00D8   31   2CF4  8719 3$:      BRW     120$                    ; Else, return
                     2CF7  8720
54   0180 C5    D0   2CF7  8721 5$:      MOVL    UCB$L_XE_CDB(R5),R4     ; Get CDB address
     56   66    D0   2CFC  8722          MOVL    P2B_L_POINTER(R6),R6    ; Point to start of data
  58   32 A3    3C   2CFF  8723          MOVZWL  IRP$W_BCNT(R3),R8       ; Get size of P2 buffer
                     2D03  8724 ;
                     2D03  8725 ; Loop to get next parameter from P2 buffer
                     2D03  8726 ;
     58   02    C2   2D03  8727 10$:     SUBL    #2,R8                   ; Try to get next parameter
          EC    19   2D06  8728          BLSS    3$                      ; Br if not there
     50   86    3C   2D08  8729          MOVZWL  (R6)+,R0                ; Get parameter type from P2
          79    13   2D0B  8730          BEQL    90$                     ; Br if null value parameter
     57   5A    D0   2D0D  8731          MOVL    R10,R7                  ; Get verification table address
0B0E 8F   50    B1   2D10  8732          CMPW    R0,#NMA$C_PCLI_PTY      ; Is this the protocol type?
          04    12   2D15  8733          BNEQ    20$                     ; Br if not
          04    A8   2D17  8734          BISW    #UCB$M_XE_PROTYP,-      ; Indicate that protocol type specified
        68  A5         2D19  8735                  UCB$W_DEVSTS(R5)        ;
                     2D1B  8736 ;
                     2D1B  8737 ; Loop to store buffer parameter in UCB/CDB
                     2D1B  8738 ;
                     2D1B  8739          ASSUME  PRM_W_TYPE EQ 0
     51   87    3C   2D1B  8740 20$:     MOVZWL  (R7)+,R1                ; Get parameter type code
          66    13   2D1E  8741          BEQL    90$                     ; Br if end of verify table
51   F000 8F    AA   2D20  8742          BICW    #^C<PRM_TYP_M_CODE>,R1  ; Clear all but type code
                     2D25  8743          ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
     59   87    9A   2D25  8744          MOVZBL  (R7)+,R9                ; Get flags byte
     51   50    B1   2D28  8745          CMPW    R0,R1                   ; Parameters match?
          17    13   2D2B  8746          BEQL    30$                     ; Br if yes
```

```
                57   02   C0  2D2D  8747          ADDL    #2,R7                           ; Skip offset word
                                2D30  8748          SKIP    PRM_FLG_V_MIN,R9,R7             ; Skip minimum value
                                2D36  8749          SKIP    PRM_FLG_V_MAX,R9,R7             ; Skip maximum value
                                2D3C  8750          SKIP    PRM_FLG_V_INVALID,R9,R7         ; Skip invalid flags
                     D7   11    2D42  8751          BRB     20$                             ; Try next parameter
                                2D44  8752 ;
                                2D44  8753 ; Match found - nullify if same value & check min,max,valid,invalid
                                2D44  8754 ;
                51   87   B0    2D44  8755 30$:    MOVW    (R7)+,R1                        ; Get offset + width
                     0A   EF    2D47  8756         EXTZV   #PRM_OFF_V_WIDTH,-              ; Get width only
           52   51   06         2D49  8757                 #PRM_OFF_S_WIDTH,R1,R2          ;   ...
                     00   EF    2D4C  8758         EXTZV   #PRM_OFF_V_VALUE,-              ; Get offset only
           51   51   0A         2D4E  8759                 #PRM_OFF_S_VALUE,R1,R1          ;   ...
             05    59   03    E0  2D51  8760         BBS     #PRM_FLG_V_CDB,R9,40$           ; Br if CDB datum
                51   55   C0    2D55  8761         ADDL    R5,R1                           ; Compute offset in UCB
                     03   11    2D58  8762         BRB     50$                             ; Continue
                51   54   C0    2D5A  8763 40$:    ADDL    R4,R1                           ; Compute offset in CDB
                                2D5D  8764 50$:    ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
        2E FB   A7   0C    E0  2D5D  8765         BBS     #PRM_TYP_V_STRING,-5(R7),100$ ; Br if string data
                58   04    C2  2D62  8766         SUBL    #4,R8                           ; Can we get value?
                     68   19  2D65  8767         BLSS    120$                            ; Br if no - exit
                53   86   D0    2D67  8768         MOVL    (R6)+,R3                        ; Get parameter value
                                2D6A  8769         CASE    R2,TYPE=B,LIMIT=#1,<-           ; Br to handler
                                2D6A  8770                 60$,-                           ;   Byte value
                                2D6A  8771                 70$,-                           ;   Word value
                                2D6A  8772                 80$>                            ;   Longword value
                                2D74  8773 ;
                                2D74  8774 ; Byte, word, longword value in structure
                                2D74  8775 ;
                61   53   90    2D74  8776 60$:    MOVB    R3,(R1)                         ; Store byte value
                   FF89   31    2D77  8777 65$:    BRW     10$                             ; Check remainder
                61   53   B0    2D7A  8778 70$:    MOVW    R3,(R1)                         ; Store word value
                   FF83   31    2D7D  8779         BRW     10$                             ; Check remainder
                61   55   D0    2D80  8780 80$:    MOVL    R5,(R1)                         ; Store longword value
                   FF7D   31    2D83  8781         BRW     10$                             ; See if more left in P2 buffer
                                2D86  8782 ;
                                2D86  8783 ; Unknown or invalidated parameter
                                2D86  8784 ;
                56   04   C0    2D86  8785 90$:    ADDL    #4,R6                           ; Skip value parameter
                58   04   C2    2D89  8786         SUBL    #4,R8                           ; Assume a value parameter
                     E9   14    2D8C  8787         BGTR    65$                             ; Br if more
                     3F   11    2D8E  8788         BRB     120$                            ; Else, all done
                                2D90  8789 ;
                                2D90  8790 ; String parameter in structure
                                2D90  8791 ;
                58   02   C2    2D90  8792 100$:   SUBL    #2,R8                           ; Can we get string?
                     3A   19    2D93  8793         BLSS    120$                            ; Br if no - exit
                59   86   3C    2D95  8794         MOVZWL  (R6)+,R9                        ; Get string length
                58   59   C2    2D98  8795         SUBL    R9,R8                           ; Can we read entire string?
                     32   19    2D9B  8796         BLSS    120$                            ; Br if no - exit
           50   0B21   8F   B1    2D9D  8797         CMPW    #NMA$C_PCLI_DES,R0              ; Is this the destination address?
                     05   12    2DA2  8798         BNEQ    101$                            ; Br if no
                   0244   30    2DA4  8799         BSBW    SET_DESAD                       ; Else, set new destination address
                     20   11    2DA7  8800         BRB     110$                            ; Continue
           50   0B04   8F   B1    2DA9  8801 101$:   CMPW    #NMA$C_PCLI_PHA,R0              ; Is this the physical address?
                     05   12    2DAE  8802         BNEQ    103$                            ; Br if no
                   022D   30    2DB0  8803         BSBW    SET_PHYAD                       ; Else set new physical address
```

```
                     14   11   2DB3  8804            BRB     110$                            ; Continue
50   0B0F 8F         B1   2DB5  8805 103$:   CMPW    #NMA$C_PCLI_MCA,R0              ; Is this the multicast address list?
                     05   12   2DBA  8806            BNEQ    105$                            ; Br if no
                   0154   30   2DBC  8807            BSBW    SET_MULTI                       ; Else, set up new UCB multicast list
                     08   11   2DBF  8808            BRB     110$                            ; Continue
                     3E   BB   2DC1  8809 105$:   PUSHR   #^M<R1,R2,R3,R4,R5>      ; Save registers
61   66   59         28   2DC3  8810            MOVC3   R9,(R6),(R1)                    ; Store string
                     3E   BA   2DC7  8811            POPR    #^M<R1,R2,R3,R4,R5>      ; Restore registers
     56   59         C0   2DC9  8812 110$:   ADDL    R9,R6                           ; Point past the string in P2 buffer
          FF34       31   2DCC  8813            BRW     10$                             ; Try for more in P2 buffer
                          2DCF  8814
     07DE 8F         BA   2DCF  8815 120$:   POPR    #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Restore registers
                     05   2DD3  8816            RSB                                     ; Return to caller
```

```
                        2DD4  8818          .SBTTL  RETURN_P2,  Return UCB/CDB buffer parameters
                        2DD4  8819
                        2DD4  8820 ;++
                        2DD4  8821 ; RETURN_P2 - Return P2 buffer parameters
                        2DD4  8822 ;
                        2DD4  8823 ; This routine is called to return the UCB/CDB buffer parameters.
                        2DD4  8824 ;
                        2DD4  8825 ; Inputs:
                        2DD4  8826 ;
                        2DD4  8827 ;       R3 = IRP address
                        2DD4  8828 ;       R5 = UCB address
                        2DD4  8829 ;
                        2DD4  8830 ; Implicit inputs:
                        2DD4  8831 ;
                        2DD4  8832 ;       IRP$L_XE_P2BUF(R3) = User P2 buffer address
                        2DD4  8833 ;       IRP$W_XE_USERSIZ(R3) = User P2 buffer size
                        2DD4  8834 ;
                        2DD4  8835 ; Outputs:
                        2DD4  8836 ;
                        2DD4  8837 ;       R0 = Size of buffer returned
                        2DD4  8838 ;       All other registers are preserved.
                        2DD4  8839 ;
                        2DD4  8840 ;--
                        2DD4  8841
                        2DD4  8842 RETURN_P2::                             ; Return P2 buffer parameters
    03DE 8F     BB      2DD4  8843          PUSHR   #^M<R1,R2,R3,R4,R6,R7,R8,R9> ; Save registers
         50     D4      2DD8  8844          CLRL    R0                     ; Assume no P2 buffer given
56   40 A3     D0      2DDA  8845          MOVL    IRP$L_XE_P2BUF(R3),R6  ; Get user P2 buffer address
       03     12      2DDE  8846          BNEQ    5$                     ; Br if given
     00B7     31      2DE0  8847          BRW     70$                    ; Else, return
                      2DE3  8848
54   0180 C5    D0      2DE3  8849 5$:      MOVL    UCB$L_XE_CDB(R5),R4    ; Get CDB address
  58   38 A3     3C      2DE8  8850          MOVZWL  IRP$W_XE_USERSIZ(R3),R8 ; Get size of user buffer
         56     DD      2DEC  8851          PUSHL   R6                     ; Save start of data address
         53     DD      2DEE  8852          PUSHL   R3                     ; Save IRP address
51   D29B CF    9E      2DF0  8853          MOVAB   LINE_PARAM,R1          ; Get address of verification talbe
                      2DF5  8854 ;
                      2DF5  8855 ; Loop to return next parameter
                      2DF5  8856 ;
                      2DF5  8857          ASSUME  PRM_W_TYPE EQ 0
     57   81    B0      2DF5  8858 10$:     MOVW    (R1)+,R7               ; Get parameter type code
       03   12      2DF8  8859          BNEQ    11$                    ; Br if end of verify table
     0096   31      2DFA  8860          BRW     65$                    ; ...
                      2DFD  8861
59   57   F000 8F    AB      2DFD  8862 11$:     BICW3   #^C<PRM_TYP_M_CODE>,R7,R9 ; Get only the type code
     53   81    9A      2E03  8863          MOVZBL  (R1)+,R3               ; Get flags byte
     50   81    B0      2E06  8864          MOVW    (R1)+,R0               ; Get offset + width
                      2E09  8865 ;
                      2E09  8866 ; We will only return NMA$C_PCLI_DES to the SHARED-LIMITED users.
                      2E09  8867 ;
     59   0B21 8F    B1      2E09  8868          CMPW    #NMA$C_PCLI_DES,R9     ; Is this a point-to-point parameter?
       07   12      2E0E  8869          BNEQ    13$                    ; Br if not
00D4 C5    02    91      2E10  8870          CMPB    #NMA$C_ACC_LIM,UCB$B_XE_ACC(R5) ; Is this a SHARED-LIMITED user?
       43   12      2E15  8871          BNEQ    50$                    ; Br if not, else return parameter
       0A   EF      2E17  8872 13$:     EXTZV   #PRM_OFF_V_WIDTH,-     ; Get width only
52   50   06              2E19  8873                  #PRM_OFF_S_WIDTH,R0,R2 ;       ...
       00   EF      2E1C  8874          EXTZV   #PRM_OFF_V_VALUE,-     ; Get offset only
```

```
                50    50    0A        2E1E  8875                    #PRM_OFF_S_VALUE,R0,R0  ;
                  05  53    03    E0  2E21  8876            BBS     #PRM_FLG_V_CDB,R3,15$   ; Br if CDB datum
                      50    55    C0  2E25  8877            ADDL    R5,R0                   ; Compute offset in UCB
                            07    11  2E28  8878            BRB     17$                     ; Continue
                            54    D5  2E2A  8879 15$:       TSTL    R4                      ; Is CDB given?
                            2C    13  2E2C  8880            BEQL    50$                     ; Br if no
                      50    54    C0  2E2E  8881            ADDL    R4,R0                   ; Compute offset in CDB
                      58    02    C2  2E31  8882 17$:       SUBL    #2,R8                   ; Any room left in buffer?
                            54    19  2E34  8883            BLSS    60$                     ; Br if no - all done
                      86    57    B0  2E36  8884            MOVW    R7,(R6)+                ; Return parameter
                30 FB A1    0C    E0  2E39  8885            BBS     #PRM_TYP_V_STRING,-5(R1),55$ ; Br if string parameter
                      58    04    C2  2E3E  8886            SUBL    #4,R8                   ; Any room left?
                            47    19  2E41  8887            BLSS    60$                     ; Br if no - all done
                                      2E43  8888            CASE    R2,TYPE=B,LIMIT=# ,<-   ; Br to handler
                                      2E43  8889                    20$,-                   ;   Byte value
                                      2E43  8890                    30$,-                   ;   Word value
                                      2E43  8891                    40$>                    ;   Longword value
                                      2E4D  8892 ;
                                      2E4D  8893 ; Byte, word, longword value in structure
                                      2E4D  8894 ;
                      86    60    9A  2E4D  8895 20$:       MOVZBL  (R0),(R6)+              ; Store byte value
                            08    11  2E50  8896            BRB     50$                     ;
                      86    60    3C  2E52  8897 30$:       MOVZWL  (R0),(R6)+              ; Store word value
                            03    11  2E55  8898            BRB     50$                     ;
                      86    60    D0  2E57  8899 40$:       MOVL    (R0),(R6)+              ; Store longword value
                                      2E5A  8900 50$:       SKIP    PRM_FLG_V_MIN,R3,R1     ; Skip minimum value
                                      2E60  8901            SKIP    PRM_FLG_V_MAX,R3,R1     ; Skip maximum value
                                      2E66  8902            SKIP    PRM_FLG_V_INVALID,R3,R1 ; Skip invalid flags
                            87    11  2E6C  8903            BRB     10$                     ; Try for more parameters
                                      2E6E  8904 ;
                                      2E6E  8905 ; String value in structure
                                      2E6E  8906 ;
                59    0B0F  8F    B1  2E6E  8907 55$:       CMPW    #NMA$C_PCLI_MCA,R9      ; Is this the multicast address list?
                            05    12  2E73  8908            BNEQ    57$                     ; Br if no
                               01C7   30  2E75  8909            BSBW    RETURN_MULTI            ; Else, return multicast address list
                            E0    11  2E78  8910            BRB     50$                     ; Try for more parameters
                      58    08    C2  2E7A  8911 57$:       SUBL    #8,R8                   ; Any room left?
                            0B    19  2E7D  8912            BLSS    60$                     ; Br if no - all done
                      86    06    9B  2E7F  8913            MOVZBW  #6,(R6)+                ; Store string size
                      86    80    D0  2E82  8914            MOVL    (R0)+,(R6)+             ; Move data
                      86    80    B0  2E85  8915            MOVW    (R0)+,(R6)+             ;      ...
                            D0    11  2E88  8916            BRB     50$                     ; Try for more parameters
                                      2E8A  8917
                      53    6E    D0  2E8A  8918 60$:       MOVL    (SP),R3                 ; Get IRP address
             3A A3    0601  8F    B0  2E8D  8919            MOVW    #SS$_BUFFEROVF,IRP$W_XE_STATUS(R3) ; Return error status
                            53 8ED0   2E93  8920 65$:       POPL    R3                      ; Pop stack
                50    56    8E    C3  2E96  8921            SUBL3   (SP)+,R6,R0             ; Return size of parameters
                      03DE  8F    BA  2E9A  8922 70$:       POPR    #^M<R1,R2,R3,R4,R6,R7,R8,R9> ; Restore registers
                                  05  2E9E  8923            RSB                             ; Return to caller
```

```
                     2E9F  8925          .SBTTL  VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
                     2E9F  8926
                     2E9F  8927 ;++
                     2E9F  8928 ; VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
                     2E9F  8929 ;
                     2E9F  8930 ; Functional description:
                     2E9F  8931 ;
                     2E9F  8932 ; This routine checks all address in the multicast address list to make sure
                     2E9F  8933 ; that the logical address bit (lsb) is on.
                     2E9F  8934 ;
                     2E9F  8935 ; Inputs:
                     2E9F  8936 ;
                     2E9F  8937 ;       R3 = Size of multicast string list
                     2E9F  8938 ;       R4 = CDB address
                     2E9F  8939 ;       R5 = UCB address
                     2E9F  8940 ;       R6 = Address past multicast strings
                     2E9F  8941 ;
                     2E9F  8942 ; Outputs:
                     2E9F  8943 ;
                     2E9F  8944 ;       R0 = Low bit clear if invalid address in list
                     2E9F  8945 ;       All other registers are preserved.
                     2E9F  8946 ;
                     2E9F  8947 ;--
                     2E9F  8948
                     2E9F  8949 VALID_MULTI:                            ; Validate the multicast address list
004C 8F    BB        2E9F  8950          PUSHR   #^M<R2,R3,R6>          ; Save some registers
50   01    CE        2EA3  8951          MNEGL   #1,R0                  ; Assume success
56   53    C2        2EA6  8952          SUBL    R3,R6                  ; Point back at start of list
53   02    C2        2EA9  8953          SUBL    #2,R3                  ; Can we read modifier word?
     26    19        2EAC  8954          BLSS    20$                    ; Br if no - error
                     2EAE  8955 ;
                     2EAE  8956 ; Make sure modifier word is valid - non-zero and less than or equal to
                     2EAE  8957 ; NMA$C_LINMC_CAL
                     2EAE  8958 ;
                     2EAE  8959          ASSUME  NMA$C_LINMC_SET EQ 1
                     2EAE  8960          ASSUME  NMA$C_LINMC_CLR EQ 2
                     2EAE  8961          ASSUME  NMA$C_LINMC_CAL EQ 3
52   86    B0        2EAE  8962          MOVW    (R6)+,R2               ; Get modifier value
     21    13        2EB1  8963          BEQL    20$                    ; Br if zero - illegal
03   52    B1        2EB3  8964          CMPW    R2,#NMA$C_LINMC_CAL    ; Is the modifier okay?
     1C    1A        2EB6  8965          BGTRU   20$                    ; Br if no - error
     1C    13        2EB8  8966          BEQL    30$                    ; Br if "CLEAR ALL" - ignore strings
53   06    C6        2EBA  8967          DIVL    #6,R3                  ; Calculate number of strings
     17    13        2EBD  8968          BEQL    30$                    ; Br if none
  12 66    E9        2EBF  8969 10$:     BLBC    (R6),20$               ; Br if not a logical address
50   86    D1        2EC2  8970          CMPL    (R6)+,R0               ; Do low order 32 bits = -1?
     05    12        2EC5  8971          BNEQ    15$                    ; Br if no - okay
50   66    B1        2EC7  8972          CMPW    (R6),R0                ; Do high order 16 bits = -1?
     08    13        2ECA  8973          BEQL    20$                    ; Br if yes - illegal
56   02    C0        2ECC  8974 15$:     ADDL    #2,R6                  ; Point to next multicast address
  ED 53    F5        2ECF  8975          SOBGTR  R3,10$                 ; Loop if more
     02    11        2ED2  8976          BRB     30$                    ; Exit with success
                     2ED4  8977
     50    D4        2ED4  8978 20$:     CLRL    R0                     ; Return error
004C 8F    BA        2ED6  8979 30$:     POPR    #^M<R2,R3,R6>          ; Restore registers
           05        2EDA  8980          RSB
```

```
                     2EDB  8982          .SBTTL  VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
                     2EDB  8983
                     2EDB  8984 ;++
                     2EDB  8985 ; VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
                     2EDB  8986 ;
                     2EDB  8987 ; Functional description:
                     2EDB  8988 ;
                     2EDB  8989 ; This routine checks the physical address to make sure the LSB is clear and
                     2EDB  8990 ; that the modifier word is valid.
                     2EDB  8991 ;
                     2EDB  8992 ; Inputs:
                     2EDB  8993 ;
                     2EDB  8994 ;       R1 = Parameter type code
                     2EDB  8995 ;       R3 = Size of string
                     2EDB  8996 ;       R4 = CDB address
                     2EDB  8997 ;       R5 = UCB address
                     2EDB  8998 ;       R6 = Address past physical address string
                     2EDB  8999 ;
                     2EDB  9000 ; Outputs:
                     2EDB  9001 ;
                     2EDB  9002 ;       R0 = Low bit clear if invalid address in list
                     2EDB  9003 ;       All other registers are preserved.
                     2EDB  9004 ;
                     2EDB  9005 ;--
                     2EDB  9006
                     2EDB  9007 VALID_PHYAD:                            ; Validate the physical address
004C 8F    BB        2EDB  9008          PUSHR   #^M<R2,R3,R6>          ; Save some registers
50   01    CE        2EDF  9009          MNEGL   #1,R0                  ; Assume success
56   53    C2        2EE2  9010          SUBL    R3,R6                  ; Point back at start of list
53   02    C2        2EE5  9011          SUBL    #2,R3                  ; Can we read modifier word?
     22    19        2EE8  9012          BLSS    30$                    ; Br if no - error
                     2EEA  9013
                     2EEA  9014 ;
                     2EEA  9015 ; Make sure modifier word is valid.
                     2EEA  9016 ;
                     2EEA  9017
                     2EEA  9018          ASSUME  NMA$C_LINMC_SET EQ 1
                     2EEA  9019          ASSUME  NMA$C_LINMC_CLR EQ 2
                     2EEA  9020          ASSUME  NMA$C_LINMC_CAL EQ 3
                     2EEA  9021          ASSUME  NMA$C_LINMC_SDF EQ 4
52   86    3C        2EEA  9022          MOVZWL  (R6)+,R2               ; Get modifier value
                     2EED  9023          $DISPATCH       R2,TYPE=B,-    ; Dispatch on modifier value
                     2EED  9024                  <-
                     2EED  9025                  <NMA$C_LINMC_SET 20$>,- ; Set the address
                     2EED  9026                  <NMA$C_LINMC_CLR 40$>,- ; Clear the address
                     2EED  9027                  <NMA$C_LINMC_CAL 30$>,- ; 3 - invalid value
                     2EED  9028                  <NMA$C_LINMC_SDF 10$>,- ; 4 - check it out more
                     2EED  9029                  >
     11    11        2EF9  9030          BRB     30$                    ; Any other values are invalid
                     2EFB  9031
51   0B04  8F    B1  2EFB  9032 10$:     CMPW    #NMA$C_PCLI_PHA,R1     ; Set to def physical addr requested?
     0A    12        2F00  9033          BNEQ    30$                    ; Return failure if not
     0A    11        2F02  9034          BRB     40$                    ; Else, success
                     2F04  9035
53   06    D1        2F04  9036 20$:     CMPL    #6,R3                  ; Is string size okay?
     03    12        2F07  9037          BNEQ    30$                    ; Br if not
  02 66    E9        2F09  9038          BLBC    (R6),40$               ; Br if a physical address
```

```
                2F0C  9039
     50   D4    2F0C  9040 30$:    CLRL    R0                      ; Return error
004C 8F   BA    2F0E  9041 40$:    POPR    #^M<R2,R3,R6>           ; Restore registers
          05    2F12  9042         RSB
```

```
                        2F13  9044          .SBTTL  SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
                        2F13  9045
                        2F13  9046 ;++
                        2F13  9047 ; SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
                        2F13  9048 ;
                        2F13  9049 ; Functional description:
                        2F13  9050 ;
                        2F13  9051 ; This routine sets up the multicast addresses in the UCB.
                        2F13  9052 ;
                        2F13  9053 ; Inputs:
                        2F13  9054 ;
                        2F13  9055 ;       R4 = CDB address
                        2F13  9056 ;       R5 = UCB address
                        2F13  9057 ;       R6 = Address of multicast addresses to be set or cleared
                        2F13  9058 ;       R9 = Size of multicast list
                        2F13  9059 ;
                        2F13  9060 ;       IPL = FIPL
                        2F13  9061 ;
                        2F13  9062 ; Outputs:
                        2F13  9063 ;
                        2F13  9064 ;       R0 = Status return for request
                        2F13  9065 ;       All registers are preserved.
                        2F13  9066 ;
                        2F13  9067 ;--
                        2F13  9068
                        2F13  9069 SET_MULTI:                              ; Set up the UCB multicast address list
024E 8F     BB          2F13  9070          PUSHR   #^M<R1,R2,R3,R6,R9>    ; Save registers
59   02     C2          2F17  9071          SUBL    #2,R9                  ; Can we read the modifier word?
       65   19          2F1A  9072          BLSS    90$                    ; Br if no - exit
59   06     C6          2F1C  9073          DIVL    #6,R9                  ; Calculate number of addresses
51   86     3C          2F1F  9074          MOVZWL  (R6)+,R1               ; Get the modifier
                        2F22  9075          ASSUME  NMA$C_LINMC_SET EQ 1
                        2F22  9076          ASSUME  NMA$C_LINMC_CLR EQ 2
                        2F22  9077          ASSUME  NMA$C_LINMC_CAL EQ 3
                        2F22  9078          CASE    R1,TYPE=B,LIMIT=#1,<-  ; Dispatch on modifier
                        2F22  9079                  10$,-                  ; Set the address(es)
                        2F22  9080                  40$,-                  ; Clear the address(es)
                        2F22  9081                  70$>                   ; Clear ALL addresses
                        2F2C  9082 ;
                        2F2C  9083 ; Set address from list
                        2F2C  9084 ;
     59     D5          2F2C  9085 10$:     TSTL    R9                     ; Any addresses present?
       51   13          2F2E  9086          BEQL    90$                    ; Br if no - exit
51   86     D0          2F30  9087 20$:     MOVL    (R6)+,R1               ; Get multicast address
52   86     B0          2F33  9088          MOVW    (R6)+,R2               ;       ...
  0169      30          2F36  9089          BSBW    MATCH_ADDRESS          ; Try to find address in table
  0F 50     E8          2F39  9090          BLBS    R0,30$                 ; Br if present - skip it
     4B     10          2F3C  9091          BSBB    FIND_MLTENTRY          ; Find entry in UCB multicast table
  43 50     E9          2F3E  9092          BLBC    R0,100$                ; Br if none - leave in error
83   51     D0          2F41  9093          MOVL    R1,(R3)+               ; Insert new address
83   52     B0          2F44  9094          MOVW    R2,(R3)+               ;       ...
00E7 C5     96          2F47  9095          INCB    UCB$B_XE_MULTI(R5)     ; Count one more address
  E2 59     F5          2F4B  9096 30$:     SOBGTR  R9,20$                 ; Br if more
     31     11          2F4E  9097          BRB     90$                    ; All done
                        2F50  9098 ;
                        2F50  9099 ; Clear address from list
                        2F50  9100 ;
```

```
                59   D5  2F50  9101 40$:    TSTL    R9                      ; Any addresses present?
                2D   13  2F52  9102         BEQL    90$                     ; Br if no - exit
            51  86   D0  2F54  9103 50$:    MOVL    (R6)+,R1                ; Get multicast address
            52  86   B0  2F57  9104         MOVW    (R6)+,R2                ;     ...
              0145   30  2F5A  9105         BSBW    MATCH_ADDRESS           ; Try to find address in table
              0B 50  E9  2F5D  9106         BLBC    R0,60$                  ; Br if not present - skip it
                83   D4  2F60  9107         CLRL    (R3)+                   ; Mark slot as not in use
                83   B4  2F62  9108         CLRW    (R3)+                   ;     ...
          00E7 C5  97  2F64  9109         DECB    UCB$B_XE_MULTI(R5)      ; Count one less address
              0042   30  2F68  9110         BSBW    SQUEEZ_MULTI            ; Squeeze up the multicast list
              E6 59  F5  2F6B  9111 60$:    SOBGTR  R9,50$                  ; Br if more
                11   11  2F6E  9112         BRB     90$                     ; All done
                         2F70  9113 ;
                         2F70  9114 ; Clear all multicast addresses
                         2F70  9115 ;
          00E7 C5  94  2F70  9116 70$:    CLRB    UCB$B_XE_MULTI(R5)      ; Reset number of multicast addresses
            51   1E  9A  2F74  9117         MOVZBL  #MAX_C_MCT*3,R1         ; Get number of words in multicast list
        52  00E9 C5  9E  2F77  9118         MOVAB   UCB$G_XE_MULTI(R5),R2   ; Get address of multicast addresses
                82   B4  2F7C  9119 80$:    CLRW    (R2)+                   ; Init multicast address list
              FB 51  F5  2F7E  9120         SOBGTR  R1,80$                  ; Loop if more
                         2F81  9121
            50   01  9A  2F81  9122 90$:    MOVZBL  S^#SS$_NORMAL,R0        ; Return success
          024E 8F   BA  2F84  9123 100$:   POPR    #^M<R1,R2,R3,R6,R9>     ; Restore registers
                05   2F88  9124         RSB                             ; Return to caller
                         2F89  9125
                         2F89  9126 ;+
                         2F89  9127 ; FIND_MLTENTRY - FIND EMPTY SLOT IN UCB MULTICAST ADDRESS LIST
                         2F89  9128 ;
                         2F89  9129 ; Inputs:
                         2F89  9130 ;
                         2F89  9131 ;       R5 = UCB address
                         2F89  9132 ;
                         2F89  9133 ; Outputs:
                         2F89  9134 ;
                         2F89  9135 ;       R0 = Status return for request
                         2F89  9136 ;       R3 = Address of available slot if successful
                         2F89  9137 ;-
                         2F89  9138 FIND_MLTENTRY:
                51   DD  2F89  9139         PUSHL   R1                      ; Save R1
            50  01   CE  2F8B  9140         MNEGL   #1,R0                   ; Assume success
        53  00E9 C5  9E  2F8E  9141         MOVAB   UCB$G_XE_MULTI(R5),R3   ; Get address of multicast list
            51   0A  9A  2F93  9142         MOVZBL  #MAX_C_MCT,R1           ; Get maximum number of addresses
                83   D5  2F96  9143 10$:    TSTL    (R3)+                   ; Empty slot?
                04   12  2F98  9144         BNEQ    20$                     ; Br if no - skip to next entry
                63   B5  2F9A  9145         TSTW    (R3)                    ; Really?
                08   13  2F9C  9146         BEQL    30$                     ; Br if yes - success
            53  02   C0  2F9E  9147 20$:    ADDL    #2,R3                   ; Skip to next address
              F2 51   F5  2FA1  9148         SOBGTR  R1,10$                  ; Br if more to try
                         2FA4  9149
                50   D4  2FA4  9150         CLRL    R0                      ; Return failure
            53  04   C2  2FA6  9151 30$:    SUBL    #4,R3                   ; Back up pointer
                51 8ED0  2FA9  9152         POPL    R1                      ; Restore registers
                   05   2FAC  9153         RSB
                         2FAD  9154
                         2FAD  9155 ;+
                         2FAD  9156 ; SQUEEZ_MULTI - SQUEEZE UP THE MULTICAST ADDRESS LIST
                         2FAD  9157 ;
```

```
                        2FAD  9158 ; Functional description:
                        2FAD  9159 ;
                        2FAD  9160 ; The first empty slot is searched for in the multicast address list and
                        2FAD  9161 ; the list is squeezed from that point down.
                        2FAD  9162 ;
                        2FAD  9163 ; Inputs:
                        2FAD  9164 ;
                        2FAD  9165 ;       R5 = UCB address
                        2FAD  9166 ;
                        2FAD  9167 ; Ouputs:
                        2FAD  9168 ;
                        2FAD  9169 ;       None
                        2FAD  9170 ;
                        2FAD  9171 ;--
                        2FAD  9172 SQUEEZ_MULTI:                                ; Squeeze down the multicast address list
                        2FAD  9173         PUSHQ   R3                           ; Save R3, R4
54   00E7 C5   9A       2FB0  9174         MOVZBL  UCB$B_XE_MULTI(R5),R4        ; Get number of entries in list
          25   13       2FB5  9175         BEQL    40$                          ; Br if none
53   00E9 C5   9E       2FB7  9176         MOVAB   UCB$G_XE_MULTI(R5),R3        ; Get address of multicast list
                        2FBC  9177 ;
                        2FBC  9178 ; Search for empty slot in list
                        2FBC  9179 ;
          63   D5       2FBC  9180 10$:    TSTL    (R3)                         ; Is this the empty slot
          05   12       2FBE  9181         BNEQ    20$                          ; Br if no
       04 A3   B5       2FC0  9182         TSTW    4(R3)                        ; Really?
          08   13       2FC3  9183         BEQL    30$                          ; Br if yes
       53 06   C0       2FC5  9184 20$:    ADDL    #6,R3                        ; Skip to next entry in list
        F1 54  F5       2FC8  9185         SOBGTR  R4,10$                       ; Loop if more possiblities
          0F   11       2FCB  9186         BRB     40$                          ; Else, exit
                        2FCD  9187 ;
                        2FCD  9188 ; Empty slot found - put the squeeze on.
                        2FCD  9189 ;
83   06 A3   D0         2FCD  9190 30$:    MOVL    6(R3),(R3)+                  ; Squeeze the list
83   06 A3   B0         2FD1  9191         MOVW    6(R3),(R3)+                  ;       ...
     F5 54   F5         2FD5  9192         SOBGTR  R4,30$                       ; Loop if more to go
        83   D4         2FD8  9193         CLRL    (R3)+                        ; Zero the last entry
        83   B4         2FDA  9194         CLRW    (R3)+                        ;       ...
                        2FDC  9195 40$:    POPQ    R3                           ; Restore R3, R4
                   05   2FDF  9196         RSB                                  ; Return to caller
```

```
                        2FE0  9198          .SBTTL  SET_PHYAD - SET THE PHYSICAL ADDRESS
                        2FE0  9199          .SBTTL  SET_DESAD - SET THE DESTINATION ADDRESS
                        2FE0  9200 ;++
                        2FE0  9201 ; SET_PHYAD - SET THE PHYSICAL ADDRESS
                        2FE0  9202 ; SET_DESAD - SET THE DESTINATION ADDRESS
                        2FE0  9203 ;
                        2FE0  9204 ; Functional description:
                        2FE0  9205 ;
                        2FE0  9206 ; This routine sets up the physical address in the CDB.
                        2FE0  9207 ;
                        2FE0  9208 ; Inputs:
                        2FE0  9209 ;
                        2FE0  9210 ;       R4 = CDB address
                        2FE0  9211 ;       R5 = UCB address
                        2FE0  9212 ;       R6 = Address of physical address to be set or cleared
                        2FE0  9213 ;       R9 = Size of the string
                        2FE0  9214 ;
                        2FE0  9215 ;       IPL = FIPL
                        2FE0  9216 ;
                        2FE0  9217 ; Outputs:
                        2FE0  9218 ;
                        2FE0  9219 ;       R0 = destroyed.
                        2FE0  9220 ;       All other registers are preserved.
                        2FE0  9221 ;
                        2FE0  9222 ;--
                        2FE0  9223          .ENABL  LSB
                        2FE0  9224 SET_PHYAD:                               ; Set up the CDB physical address
    02C2 8F    BB       2FE0  9225          PUSHR   #^M<R1,R6,R7,R9>        ; Save registers
57  00E1 C5    9E       2FE4  9226          MOVAB   UCB$G_XE_PHA(R5),R7     ; Get address of UCB cell
         09    11       2FE9  9227          BRB     10$                     ; Join common code
                        2FEB  9228 SET_DESAD:
    02C2 8F    BB       2FEB  9229          PUSHR   #^M<R1,R6,R7,R9>        ; Save registers
57  00CC C5    9E       2FEF  9230          MOVAB   UCB$G_XE_DES(R5),R7     ; Get address of UCB cell
    59   02    C2       2FF4  9231 10$:     SUBL    #2,R9                   ; Can we read the modifier word?
         41    19       2FF7  9232          BLSS    50$                     ; Br if no - exit
    59   06    C6       2FF9  9233          DIVL    #6,R9                   ; Calculate number of addresses
    51   86    3C       2FFC  9234          MOVZWL  (R6)+,R1                ; Get the modifier
                        2FFF  9235
                        2FFF  9236          ASSUME  NMA$C_LINMC_SET EQ 1
                        2FFF  9237          ASSUME  NMA$C_LINMC_CLR EQ 2
                        2FFF  9238          ASSUME  NMA$C_LINMC_CAL EQ 3
                        2FFF  9239          ASSUME  NMA$C_LINMC_SDF EQ 4
                        2FFF  9240          $DISPATCH       R1,TYPE=B,-     ; Dispatch on modifier value
                        2FFF  9241                  <-
                        2FFF  9242                  <NMA$C_LINMC_SET 20$>,- ; Set the address
                        2FFF  9243                  <NMA$C_LINMC_CLR 30$>,- ; Clear the address
                        2FFF  9244                  <NMA$C_LINMC_CAL 50$>,- ; 3 - invalid value
                        2FFF  9245                  <NMA$C_LINMC_SDF 40$>,- ; Set physical as DECnet default address
                        2FFF  9246                  >
         2D    11       300B  9247          BRB     50$                     ; Any other values are invalid
                        300D  9248
                        300D  9249 ;
                        300D  9250 ; Set physical address
                        300D  9251 ;
    59   D5    300D  9252 20$:     TSTL    R9                              ; Any addresses present?
         29    13    300F  9253          BEQL    50$                     ; Br if no - exit
87  86   D0    3011  9254          MOVL    (R6)+,(R7)+                     ; Set new address
```

```
                67   86   B0  3014  9255          MOVW    (R6)+,(R7)              ;       ...
                     21   11  3017  9256          BRB     50$                     ; All done
                                3019  9257 ;
                                3019  9258 ; Clear address from list
                                3019  9259 ;
                87   01   CE  3019  9260 30$:     MNEGL   #1,(R7)+                ; Reset address
                67   01   AE  301C  9261          MNEGW   #1,(R7)                 ;       ...
                     19   11  301F  9262          BRB     50$                     ; All done
                                3021  9263
                                3021  9264 ;
                                3021  9265 ; Set the physical address (Assume entered from SET_PHYAD) to the DECNET
                                3021  9266 ; default address. The physical address is pre-set to -1 to default to
                                3021  9267 ; the current PHA or HWA address. We will leave it alone if we are already
                                3021  9268 ; initialized or we don't have SCSSYSTEMID set.
                                3021  9269 ;
                     00   E0  3021  9270 40$:     BBS     #CDB_STS_V_INITED,-     ; Br if device is already inited
                13 025A C4          3023  9271                  CDB_B_STS(R4),50$  ;       ... use current PHA or HWA
           51   00000000'GF   B0  3027  9272          MOVW    G^SCS$GB_SYSTEMID,R1    ; Get unique high word of address
                     0A   13  302E  9273          BEQL    50$                     ; Br if not set, use default
           87   000400AA 8F   D0  3030  9274          MOVL    #^X000400AA,(R7)+       ; Set common low longword of addr
                67   51   B0  3037  9275          MOVW    R1,(R7)                 ; Set unique high word of address
                                303A  9276
                02C2 8F   BA  303A  9277 50$:     POPR    #^M<R1,R6,R7,R9>        ; Restore registers
                     05   303E  9278          RSB                             ; Return to caller
                                303F  9279          .DSABL  LSB
```

```
                        303F  9281          .SBTTL  RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
                        303F  9282
                        303F  9283 ;++
                        303F  9284 ; RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
                        303F  9285 ;
                        303F  9286 ; Functional description:
                        303F  9287 ;
                        303F  9288 ; This routine returns all the multicast addresses in the UCB list.
                        303F  9289 ;
                        303F  9290 ; Inputs:
                        303F  9291 ;
                        303F  9292 ;       R4 = CDB address
                        303F  9293 ;       R5 = UCB address
                        303F  9294 ;       R6 = Address where to store multicast address list
                        303F  9295 ;       R8 = Size left in output buffer
                        303F  9296 ;
                        303F  9297 ; Outputs:
                        303F  9298 ;
                        303F  9299 ;       R6 = Address past stored multicast address list
                        303F  9300 ;       All other registers are preserved.
                        303F  9301 ;
                        303F  9302 ;--
                        303F  9303
                        303F  9304 RETURN_MULTI:                           ; Return the multicast address list
                        303F  9305          PUSHQ   R1                     ; Save R1, R2
52   00E7 C5    9A      3042  9306          MOVZBL  UCB$B_XE_MULTI(R5),R2  ; Set number of multicast addresses
51   00E9 C5    9E      3047  9307          MOVAB   UCB$G_XE_MULTI(R5),R1  ; Point to start of multicast lists
     58   02    C2      304C  9308          SUBL    #2,R8                  ; Can string size fit?
          1E    19      304F  9309          BLSS    40$                    ; Br if no - return
     56   02    C0      3051  9310          ADDL    #2,R6                  ; Skip size field of return data
          56    DD      3054  9311          PUSHL   R6                     ; Save start address of string
     58   06    C2      3056  9312 10$:     SUBL    #6,R8                  ; Room left in buffer?
          09    19      3059  9313          BLSS    30$                    ; Br if no .. exit now
     86   81    D0      305B  9314          MOVL    (R1)+,(R6)+            ; Return address
     86   81    B0      305E  9315          MOVW    (R1)+,(R6)+            ;      ...
       F2 52    F5      3061  9316          SOBGTR  R2,10$                 ; Br if more possibles
                        3064  9317
          51 8ED0       3064  9318 30$:     POPL    R1                     ; Restore string address
52   56   51    C3      3067  9319          SUBL3   R1,R6,R2               ; Get string size
  FE A1   52    B0      306B  9320          MOVW    R2,-2(R1)              ; Return string size
                        306F  9321 40$:     POPQ    R1                     ; Restore R1, R2
                  05    3072  9322          RSB                            ; Return to caller
```

```
                                                                B 14
                     3073  9324          .SBTTL  MATCH_MULTI - CHECK MULTICAST ADDRESS
                     3073  9325
                     3073  9326 ;++
                     3073  9327 ; MATCH_MULTI - CHECK MULTICAST ADDRESS
                     3073  9328 ;
                     3073  9329 ; Functional description:
                     3073  9330 ;
                     3073  9331 ; This routine returns success if the unit is in promiscous mode, or the
                     3073  9332 ; recognizes all multicast address or the multicast address in the buffer
                     3073  9333 ; matches a multicast address in the unit's multicast address list.
                     3073  9334 ;
                     3073  9335 ; Inputs:
                     3073  9336 ;
                     3073  9337 ;       R2 = Receive buffer
                     3073  9338 ;       R4 = CDB address
                     3073  9339 ;       R5 = UCB address
                     3073  9340 ;
                     3073  9341 ; Outputs:
                     3073  9342 ;
                     3073  9343 ;       R0 = Status return for request
                     3073  9344 ;       R1 = Destroyed
                     3073  9345 ;       All other registers are preserved.
                     3073  9346 ;
                     3073  9347 ;--
                     3073  9348
                     3073  9349 MATCH_MULTI:                            ; Find multicast addres in UCB
                     3073  9350          PUSHQ   R2                     ; Save R2, R3
          50    D4   3076  9351          CLRL    R0                     ; Assume failure
                     3078  9352
                     3078  9353          ASSUME  UCB$V_XE_INITED  EQ  0
       22 68  A5    E9   3078  9354          BLBC    UCB$W_DEVSTS(R5),10$   ; Br if PROTOCOL TYPE is not valid
              50    D6   307C  9355          INCL    R0                     ; Assume success
                         307E  9356          ASSUME  NMA$C_STATE_ON EQ 0
                         307E  9357          ASSUME  NMA$C_STATE_OFF EQ 1
       1B 00DB  C5    E9   307E  9358          BLBC    UCB$B_XE_PRM(R5),10$   ; Br if promiscuous mode - OKAY
       16 00DC  C5    E9   3083  9359          BLBC    UCB$B_XE_MLT(R5),10$   ; Br if all multicasts are enabled
       51    38 A2    7D   3088  9360          MOVQ    CXB$G_R_DEST(R2),R1    ; Get multicast address
51  FFFFFFFF  8F    D1   308C  9361          CMPL    #BRDCSTT,R1            ; Is this the broadcast address?
              07    12   3093  9362          BNEQ    5$                     ; Br if no
    52   FFFF  8F    B1   3095  9363          CMPW    #BRDCST2,R2            ; Really?
              02    13   309A  9364          BEQL    10$                    ; Br if yes - everybody get this one
              04    10   309C  9365 5$:       BSBB    MATCH_ADDRESS          ; Else, find exact match
                         309E  9366 10$:      POPQ    R2                     ; Restore R2, R3
                    05   30A1  9367          RSB                            ; Return to caller
```

```
                        30A2  9369          .SBTTL  MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
                        30A2  9370
                        30A2  9371 ;++
                        30A2  9372 ; MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
                        30A2  9373 ;
                        30A2  9374 ; Functional description:
                        30A2  9375 ;
                        30A2  9376 ; This routine searches the UCB multicast address list for a match on a
                        30A2  9377 ; multicast address.
                        30A2  9378 ;
                        30A2  9379 ; Inputs:
                        30A2  9380 ;
                        30A2  9381 ;       R1 = Low 32 bits of 48 bit multicast address to match
                        30A2  9382 ;       R2 = High 16 bits of 48 bit multicast address to match
                        30A2  9383 ;       R4 = CDB address
                        30A2  9384 ;       R5 = UCB address
                        30A2  9385 ;
                        30A2  9386 ; Outputs:
                        30A2  9387 ;
                        30A2  9388 ;       R0 = Status return for request
                        30A2  9389 ;       R3 = Address of slot in multicast address list
                        30A2  9390 ;       All other registers are preserved.
                        30A2  9391 ;
                        30A2  9392 ;--
                        30A2  9393
                        30A2  9394 MATCH_ADDRESS:                          ; Find multicast address in UCB
          54    DD      30A2  9395          PUSHL   R4                     ; Save R4
     50   01    9A      30A4  9396          MOVZBL  S^#SS$_NORMAL,R0       ; Assume success
54   00E7 C5    9A      30A7  9397          MOVZBL  UCB$B_XE_MULTI(R5),R4  ; Set number of multicast addresses
53   00E9 C5    9E      30AC  9398          MOVAB   UCB$G_XE_MULTI(R5),R3  ; Point to start of multicast lists
     51   83    D1      30B1  9399 10$:     CMPL    (R3)+,R1               ; Is this a match?
          05    12      30B4  9400          BNEQ    20$                    ; Br if no - skip to next
     52   63    B1      30B6  9401          CMPW    (R3),R2                ; Is it really?
          08    13      30B9  9402          BEQL    30$                    ; Br if yes - all done
53        02    C0      30BB  9403 20$:     ADDL    #2,R3                  ; Skip to next entry
  F0 54    F5            30BE  9404          SOBGTR  R4,10$                 ; Br if more in list
                         30C1  9405
          50    D4      30C1  9406          CLRL    R0                     ; Return failure
53        04    C2      30C3  9407 30$:     SUBL    #4,R3                  ; Backup pointer
          54 8ED0       30C6  9408          POPL    R4                     ; Restore R4
                   05   30C9  9409          RSB                            ; Return to caller
```

```
                                30CA  9411          .SBTTL  POKE_USER - DELIVER ATTENTION ASTS
                                30CA  9412
                                30CA  9413 ;++
                                30CA  9414 ; POKE_USER - Deliver attention AST
                                30CA  9415 ;
                                30CA  9416 ; Functional description:
                                30CA  9417 ;
                                30CA  9418 ; This routine is used to deliver an attention AST if one has been
                                30CA  9419 ; requested.
                                30CA  9420 ;
                                30CA  9421 ; Inputs:
                                30CA  9422 ;
                                30CA  9423 ;       R5 = UCB address
                                30CA  9424 ;
                                30CA  9425 ; Outputs:
                                30CA  9426 ;
                                30CA  9427 ;       R0 = Low bit clear only if user is not notified
                                30CA  9428 ;       R1-R3 are destroyed.
                                30CA  9429 ;
                                30CA  9430 ;--
                                30CA  9431
                                30CA  9432 POKE_USER:                                   ; Poke user process
                                30CA  9433          DSBINT  UCB$B_FIPL(R5)              ; Sync access to UCB
                          7E    30D1  9434          CLRL    -(SP)                       ; Assume failure
                51  00C0 C5    30D3  9435          MOVAB   UCB$L_XE_AST(R5),R1         ; Get AST listhead
                      61  D5    30D8  9436          TSTL    (R1)                        ; Empty?
                      1C  13    30DA  9437          BEQL    30$                         ; Branch if yes
                      6E  D6    30DC  9438          INCL    (SP)                        ; Indicate success
                      54  DD    30DE  9439          PUSHL   R4                          ; Save R4
                54    51  D0    30E0  9440          MOVL    R1,R4                       ; Copy listhead address
                51    61  D0    30E3  9441 10$:     MOVL    (R1),R1                     ; Address a block
                      07  13    30E6  9442          BEQL    20$                         ; Branch if done
                   44 A5  D0    30E8  9443          MOVL    UCB$L_DEVDEPEND(R5),-       ; Change parameter
                   1C A1        30EB  9444                  ACB$L_KAST+4(R1)            ;    return status
                      F4  11    30ED  9445          BRB     10$                         ; Continue thru AST blocks
00000000'GF       16    30EF  9446 20$:     JSB     G^COM$DELATTNAST            ; Deliver the AST's
                   54 8ED0    30F5  9447          POPL    R4                          ; Restore R4
                              30F8  9448
                   50 8ED0    30F8  9449 30$:     POPL    R0                          ; Return success indicator
                              30FB  9450          ENBINT                              ; Restore IPL
                         05   30FE  9451          RSB                                 ; Return to caller
```

```
                        30FF  9453          .SBTTL  MATCH_PROTYP - Match protocol type
                        30FF  9454          .SBTTL  MATCH_PROMTYP - Find the promiscuous user
                        30FF  9455
                        30FF  9456 ;++
                        30FF  9457 ; MATCH_PROTYP - Match protocol type
                        30FF  9458 ; MATCH_PROMTYP - Find the promiscuous user
                        30FF  9459 ;
                        30FF  9460 ; This routine checks for a match of a protocol type against that in
                        30FF  9461 ; existing UCB's.
                        30FF  9462 ;
                        30FF  9463 ; Inputs:
                        30FF  9464 ;
                        30FF  9465 ;       R1 = word of protocol type
                        30FF  9466 ;       R4 = CDB address
                        30FF  9467 ;
                        30FF  9468 ; Outputs:
                        30FF  9469 ;
                        30FF  9470 ;       R0 = LBS=> match; LBC=> no match
                        30FF  9471 ;       R5 = UCB address on success
                        30FF  9472 ;
                        30FF  9473 ;-
                        30FF  9474
                        30FF  9475 MATCH_PROTYP:                            ; Match protocol type
              50    D4  30FF  9476          CLRL    R0                      ; Assume failure
      55  00C8 C4    D0  3101  9477          MOVL    CDB_L_UCB0(R4),R5       ; Get first UCB address
              18    13  3106  9478          BEQL    20$                     ; Br if not inited - yet
        55    30 A5   D0  3108  9479 10$:    MOVL    UCB$L_LINK(R5),R5       ; Get next UCB address
              12    13  310C  9480          BEQL    20$                     ; If EQL no match
                        310E  9481          ASSUME  UCB$V_XE_INITED  EQ  0
          F6 68 A5    E9  310E  9482          BLBC    UCB$W_DEVSTS(R5),10$    ; Br if PROTOCOL TYPE is not valid
                        3112  9483
                        3112  9484          ASSUME  NMA$C_STATE_ON EQ 0
                        3112  9485          ASSUME  NMA$C_STATE_OFF EQ 1
                        3112  9486
        F1 00DB C5    E9  3112  9487          BLBC    UCB$B_XE_PRM(R5),10$    ; Skip if PROMISCUOUS user
    00CA C5     51    B1  3117  9488          CMPW    R1,UCB$W_XE_PROTYP(R5)  ; Match?
                EA    12  311C  9489          BNEQ    10$                     ; If NEQ no - loop
                50    D6  311E  9490 15$:     INCL    R0                      ; Return success
                05    05  3120  9491 20$:     RSB                             ; Done
                        3121  9492
                        3121  9493 MATCH_PROMTYP:
              50    01    9A  3121  9494          MOVZBL  #1,R0                   ; Assume success
      55  0224 C4    D0  3124  9495          MOVL    CDB_L_PRMUSER(R4),R5    ; Get PROMISCUOUS user's UCB address
                02    12  3129  9496          BNEQ    10$                     ; Br if present
                50    D4  312B  9497          CLRL    R0                      ; Else, return error
                05    05  312D  9498 10$:     RSB                             ; Return to caller
                        312E  9499
                        312E  9500
                        312E  9501 XE_END::
                        312E  9502          .END
```

```
$$$                     = 00000020 R      02        CDB_B_NEXTXMT                   00000018  G
$$$FLG                  = 00000013                  CDB_B_PAD                       0000025B  G
$$$MAP                  = 00000000                  CDB_B_PRM                       0000025D  G
$$$OFF                  = 0000001E                  CDB_B_RCVCNT                    000000B6  G
$$$SIZ                  = 00000008                  CDB_B_RCVMAP                    0000001A  G
$$$TYP                  = 00000429                  CDB_B_SETPRM                    0000025F  G
$$$WID                  = 00000002                  CDB_B_SPARE                     000000CC  G
$$OP                    = 00000002                  CDB_B_STS                       0000025A  G
ABORTIO                   00000383 R      03        CDB_B_TIM_DNI                   00000223  G
ABORTIO_BR                000002D3 R      03        CDB_B_TIM_XMT                   00000222  G
ABORT_IRP                 00000C08 R      03        CDB_B_TYPE                      0000000A  G
ABORT_PKT                 00001F34 R      03        CDB_B_UNTCNT                    00000275  G
ACB$L_KAST              = 00000018                  CDB_B_XMTCNT                    000000B7  G
ACCESS                    00000DF2 R      03        CDB_B_XMTMAP                    0000001B  G
ADDRCVLIST                000013AF RG     03        CDB_C_ABORTS                  = 00000001
ADD_MULTI                 00002ACD R      03        CDB_C_LENGTH                    000002BA  G
ALLOC_CDB                 0000236E R      03        CDB_C_MAPPED                  = 0000014C
ALLOC_FAIL                00000EAD R      03        CDB_C_QUEUES                  = 00000007
ALLOC_P2BUF               00000E0B R      03        CDB_C_SETPRM                  = 00000004
ALT_START                 0000066E RG     03        CDB_C_ZERO                      00000222  G
ASSPKT_DCH                00001FC3 RG     03        CDB_G_CUDB                      000000EA  G
ASSPKT_NODCH              00001F9C RG     03        CDB_G_HWA                       00000269  G
ATS_UBA                 = 00000001                  CDB_G_MAPPED                    000000D6  G
BAD_PARAM                 00000EA8 R      03        CDB_G_MLTTBL                    00000278  G
BAD_PARAM_TBL             000001AB R      03        CDB_G_MUDB                      0000012A  G
BLD_IRP                   000027B8 R      03        CDB_G_PCBB                      000000D6  G
BLD_IRP_CO                000027CE R      03        CDB_G_PHA                       00000263  G
BLD_STOP_IRP              0000274C R      03        CDB_G_PHYADR                    0000026F  G
BLD_STRT_IRP              0000277A R      03        CDB_G_RRING                     000001D2  G
BLK_B_SPARE               00000008                  CDB_G_SCRUDB                    00000166  G
BLK_B_TYPE                0000000A                  CDB_G_SIDUDB                    0000016E  G
BLK_C_HEADER              0000000C                  CDB_G_XRING                     000001AA  G
BLK_L_LINK                00000000                  CDB_G_XRUDB                     000000DE  G
BLK_T_DATA                0000000C                  CDB_L_ASSPKT                    000002B4  G
BLK_W_SIZE                00000008                  CDB_L_DEVDEPEND                 000000C4  G
BRDCST1                 = FFFFFFFF                  CDB_L_FPC                       0000000C  G
BRDCST2                 = 0000FFFF                  CDB_L_FQBL                      00000004  G
BUG$_NOBUFPCKT            ********    X     03        CDB_L_FQFL                      00000000  G
CAN$C_CANCEL            = 00000000                  CDB_L_FR3                       00000010  G
CAN$C_DASSGN            = 00000001                  CDB_L_FR4                       00000014  G
CANCEL                    000028A1 RG     03        CDB_L_PCBBMAP                   000000BC  G
CCB$L_UCB               = 00000000                  CDB_L_PCBBUA                    000000C0  G
CDB_B_AQUOTA              000002B8  G               CDB_L_PCSRO                     00000010  G
CDB_B_CON                 00000260  G               CDB_L_PRMUSER                   00000224  G
CDB_B_CRC                 0000025F  G               CDB_L_RCVMAP                    0000001C  G
CDB_B_DCH                 0000025C  G               CDB_L_RRADDR                    0000005C  G
CDB_B_DIAG1               000000CD  G               CDB_L_TQE                       00000228  G
CDB_B_EKO                 00000261  G               CDB_L_UCBO                      000000C8  G
CDB_B_FIPL                0000000B  G               CDB_L_XMTMAP                    0000003C  G
CDB_B_ILP                 00000262  G               CDB_L_XRADDR                    0000004C  G
CDB_B_LASTRCV             000000B5  G               CDB_Q_DONE                      0000008C  G
CDB_B_LASTXMT             000000B4  G               CDB_Q_INPUT                     00000084  G
CDB_B_MLT                 0000025E  G               CDB_Q_POST                      000000AC  G
CDB_B_MLTIBL              00000277  G               CDB_Q_QUEUES                    0000007C  G
CDB_B_MQUOTA              000002B9  G               CDB_Q_RCVBUF                    0000009C  G
CDB_B_MULTI               00000276  G               CDB_Q_RCVPND                    000000A4  G
CDB_B_NEXTRCV             00000019  G               CDB_Q_XMTPND                    00000094  G
```

```
CDB_Q_XMTREQ                    0000007C  G
CDB_STS_M_ERR                 = 00000040  G
CDB_STS_M_FORK_PEND           = 00000004  G
CDB_STS_M_FUNC                = 00000008  G
CDB_STS_M_INITED              = 00000001  G
CDB_STS_M_POWER               = 00000080  G
CDB_STS_M_RCBI                = 00000020  G
CDB_STS_M_RUN                 = 00000002  G
CDB_STS_M_TIMER               = 00000010  G
CDB_STS_V_ERR                 = 00000006  G
CDB_STS_V_FORK_PEND           = 00000002  G
CDB_STS_V_FUNC                = 00000003  G
CDB_STS_V_INITED              = 00000000  G
CDB_STS_V_POWER               = 00000007  G
CDB_STS_V_RCBI                = 00000005  G
CDB_STS_V_RUN                 = 00000001  G
CDB_STS_V_TIMER               = 00000004  G
CDB_W_BSZ                       000000B8  G
CDB_W_DIAG2                     000000CE  G
CDB_W_MODE                      00000258  G
CDB_W_QUOTA                     000000BA  G
CDB_W_SBUCTR                    000000D2  G
CDB_W_SIZE                      00000008  G
CDB_W_UBUCTR                    000000D4  G
CDB_W_UFDCTR                    000000D0  G
CHANGE_PARAM                    00002CE7  R       03
CHECKER                         00002934  R       03
CHECKPKT                        0000294E  R       03
CHECK_BUFS                      00000DCF  R       03
CHECK_DGBUFA                    00000E60  R       03
CHECK_DGBUFR                    00000E6D  R       03
CHECK_DGBUFW                    00000E75  R       03
CHECK_P1                        00000DF8  R       03
CHECK_P2                        00000DD1  R       03
CHECK_PARAM                     00000AAD  R       03
CHECK_QUOTA                     0000094E  R       03
CHECK_SHR                       00000652  R       03
CHECK_SRC                       00001D9E  R       03
CIRCUIT_PARAM                   0000011E  R       03
CIRC_CTR                        00000179  R       03
CIRC_CTR_BUFSIZ               = 00000020
CIRC_CTR_SIZE                 = 00000006
CLEANUP_SHR                     000027EB  RG      03
CLONED_UCB                      00000202  RG      03
COM$DECATTNAST                  ********    X     03
COM$DRVDEALMEM                  ********    X     03
COM$FLUSHATTNS                  ********    X     03
COM$POST                        ********    X     03
COM$SETATTNAST                  ********    X     03
CONTIN                          00001649  R       03
CONTROL_INIT                    00000201  RG      03
COPY_RCV                        00001DAB  R       03
CRB$L_AUXSTRUC                = 00000010
CRB$L_INTD                    = 00000024
CRC                             00002028  RG      03
CRCTABLE                        000001C1  R       03
CTRTAB                          00000129  R       03
CXB$B_CODE                    = 0000000B
CXB$B_R_FLAGS                   0000000B  G
CXB$B_TYPE                    = 0000000A
CXB$B_XE_FUNC                   00000020  G
CXB$B_XE_RING                   00000023  G
CXB$B_XE_SLOT                   00000022  G
CXB$C_HEADER                  = 00000048
CXB$C_TRAILER                 = 00000004
CXB$G_R_DEST                    00000038  G
CXB$G_R_SRC                     0000003E  G
CXB$L_BD                      = 00000004
CXB$L_END_ACTION              = 0000001C
CXB$L_FL                      = 00000000
CXB$L_IRP                     = 00000014
CXB$L_LINK                    = 00000010
CXB$L_SPARE0                  = 00000024
CXB$L_SPARE1                  = 00000020
CXB$L_T_IRP                     00000024  G
CXB$Q_STATION                 = 00000028
CXB$T_R_DATA                    00000038  G
CXB$T_R_USERDAT                 00000046  G
CXB$T_T_DATA                    0000003A  G
CXB$W_BCNT                    = 0000001A
CXB$W_BOFF                    = 00000018
CXB$W_LENGTH                  = 0000000C
CXB$W_R_LENERR                  00000014  G
CXB$W_R_NCHAIN                  0000001C  G
CXB$W_R_PTYPE                   00000044  G
CXB$W_R_SIZE                    00000046  G
CXB$W_SIZE                    = 00000008
CXB$W_XE_RID                    00000022  G
CXB_CHECKER                     00002971  R       03
CXB_CHECKPKT                    000029A6  R       03
DC$_SCOM                      = 00000020
DDB$L_DDT                     = 0000000C
DDB$L_UCB                     = 00000004
DEF_MODE                        00000084  RG      03
DELETE_BLOCK                    00002771  R       03
DELETE_SHR                      00002839  RG      03
DEV$M_AVL                     = 00040000
DEV$M_IDV                     = 04000000
DEV$M_NET                     = 00002000
DEV$M_ODV                     = 08000000
DEV$M_SHR                     = 00010000
DEV_TIMEOUT                     00002349  RG      03
DG_ERR                          00000CCA  R       03
DG_ERR1                         00000CC7  R       03
DIAGNOSE_FDT                    00000C97  RG      03
DIAG_B_SPARE                    0000000B
DIAG_B_TYPE                     0000000A
DIAG_C_EXTRA                  = 0000000A
DIAG_C_LENGTH                   0000004C
DIAG_G_DEST                     0000003E
DIAG_G_HWA                      00000036
DIAG_G_SRC                      00000044
DIAG_L_BUFFER                   00000004
DIAG_L_DATA                     00000000
```

```
DIAG_L_DEPEND                   00000024
DIAG_L_ERRS                     0000001C
DIAG_L_EXTRA                    00000020
DIAG_Q_FINISH                   00000014
DIAG_Q_PCBB                     0000002C
DIAG_Q_PCSR                     00000024
DIAG_Q_START                    0000000C
DIAG_T_DATA                     0000000C
DIAG_T_RDATA                    0000003E
DIAG_W_ERR                      00000034
DIAG_W_ERR2                     0000003C
DIAG_W_SIZE                     00000008
DIAG_W_TYPE                     0000004A
DNI_C_TIM                =      0000000C
DNI_TIM                  =      00000007
DONE                            000016D5 R      03
DONE_INTR                       00001806 RG     03
DPT$B_FLAGS              =      0000000D
DPT$C_LENGTH             =      00000038
DPT$C_VERSION            =      00000004
DPT$INITAB                      00000038 R      02
DPT$M_NOUNLOAD           =      00000004
DPT$REINITAB                    000000AB R      02
DPT$TAB                         00000000 R      02
DRV_DONE                        00000F06 R      03
DSC$A_POINTER            =      00000004
DT$_DELUA                =      00000019
DT$_DEUNA                =      0000000E
DYN$C_BUFIO              =      00000013
DYN$C_CDB                =      00000033
DYN$C_CRB                =      00000005
DYN$C_CXB                =      0000001B
DYN$C_DDB                =      00000006
DYN$C_DPT                =      0000001E
DYN$C_IRP                =      0000000A
DYN$C_ORB                =      00000049
DYN$C_TQE                =      0000000F
DYN$C_UCB                =      00000010
ESTEST                          00000D3E R      03
EXE$ABORTIO                     ********   X    03
EXE$ALLOCBUF                    ********   X    03
EXE$ALONONPAGED                 ********   X    03
EXE$BUFFRQUOTA                  ********   X    03
EXE$BUFQUOPRC                   ********   X    03
EXE$DEANONPAGED                 ********   X    03
EXE$FINISHIO                    ********   X    03
EXE$FORK                        ********   X    03
EXE$GL_TENUSEC                  ********   X    03
EXE$GL_UBDELAY                  ********   X    03
EXE$GQ_SYSTIME                  ********   X    03
EXE$INSTIMQ                     ********   X    03
EXE$PROBER_DSC                  ********   X    03
EXE$QIORETURN                   ********   X    03
EXE$READCHK                     ********   X    03
EXE$WRITECHK                    ********   X    03
FFI$L_DL_UCB             =      00000034
FFI$L_ERROR              =      0000001C
FFI$L_RECV_DONE          =      00000018
FFI$L_SHUT_DONE          =      00000020
FFI$L_XMIT               =      00000010
FFI$L_XMIT_DONE          =      00000014
FFI_INIT                        000002B5 RG     03
FILLRCVLIST                     000013AD RG     03
FIND_MLTENTRY                   00002F89 R      03
FIND_POINT_UCB                  00002A13 R      03
FIND_SHR                        000029D7 R      03
FINISH_RCV_FFI                  00001E33 RG     03
FINISH_RCV_IO                   00001E6E RG     03
FINISH_XMT_FFI                  00001E25 RG     03
FKB$B_FIPL               =      0000000B
FKB$B_TYPE               =      0000000A
FKB$L_FR3                =      00000010
FORK_COUNTER                    000019E5 R      03
FORK_INIT                       000019AE R      03
FORK_LDMEM                      00001AC6 R      03
FORK_PKT                        000019A4 R      03
FORK_PROC                       00001921 RG     03
FORK_RDMEM                      00001AAD R      03
FORK_RDSID                      00001ADF R      03
FORK_READST                     00001B21 R      03
FORK_STOP                       000019DC R      03
FORK_TIMER                      000012C1 R      03
FUNCTAB_LEN              =      0000004C
GET_CHAR_BUF                    00000D92 R      03
HALT                            00000D68 R      03
IDB$L_CSR                =      00000000
IDB$L_UCBLST             =      00000018
INACT_ERROR                     000005E1 R      03
INIT_C_AQUOTA            =      00000002
INIT_C_QUOTA             =      000034BC
INPUT                           000012FE RG     03
INPUT_BUFFER                    0000132D RG     03
INPUT_BUFFER1                   0000131B RG     03
INTEXIT                         000018F8 R      03
IN_CHMODE                       0000164F R      03
IN_ESTEST                       000017D7 R      03
IN_HALT                         000017E3 R      03
IN_INIT                         00001558 R      03
IN_LDMEM                        000016DC R      03
IN_LDSID                        00001746 R      03
IN_MODE                         00001638 R      03
IN_MULTI                        0000160F R      03
IN_MULTIC                       000016B9 R      03
IN_PHYAD                        000015CB R      03
IN_RCCNTS                       00001695 R      03
IN_RDCNTS                       0000169A R      03
IN_RDDPA                        000015BE R      03
IN_RDMEM                        0000170B R      03
IN_RDSID                        00001769 R      03
IN_READST                       0000178F R      03
IN_RING                         000015A4 R      03
IN_RSMODE                       0000168A R      03
IN_START                        000017A0 R      03
IN_STEST                        000017E3 R      03
```

```
IN_STOPPR                       00001795 R      03
IN_TESTD                        00001563 R      03
IN_UPST                         0000173B R      03
IO$V_ATTNAST                =   00000008
IO$V_CLR_COUNT              =   0000000A
IO$V_CTRC                   =   00000009
IO$V_NOW                    =   00000006
IO$V_RD_COUNT               =   00000008
IO$V_SHUTDOWN               =   00000007
IO$V_STARTUP                =   00000006
IO$_DIAGNOSE                =   0000001D
IO$_READLBLK                =   00000021
IO$_READPBLK                =   0000000C
IO$_READVBLK                =   00000031
IO$_SENSECHAR               =   0000001B
IO$_SENSEMODE               =   00000027
IO$_SETCHAR                 =   0000001A
IO$_SETMODE                 =   00000023
IO$_VIRTUAL                 =   0000003F
IO$_WRITELBLK               =   00000020
IO$_WRITEPBLK               =   0000000B
IO$_WRITEVBLK               =   00000030
IOC$ALOUBAMAP                   ********    X   03
IOC$CREDIT_UCB                  ********    X   03
IOC$DELETE_UCB                  ********    X   03
IOC$INITIATE                    ********    X   03
IOC$LOADUBAMAP                  ********    X   03
IOC$LOADUBAMAPA                 ********    X   03
IOC$LOADUBAMAPN                 ********    X   03
IOC$MNTVER                      ********    X   03
IOC$PURGDATAP                   ********    X   03
IOC$RELDATAP                    ********    X   03
IOC$RELMAPREG                   ********    X   03
IOC$REQDATAPNW                  ********    X   03
IOC$RETURN                      ********    X   03
IOC$WFIKPCH                     ********    X   03
IO_DONE                         00001F4B R      03
IO_DONE1                        00001F4F R      03
IO_DONE2                        00001F65 R      03
IO_DONE3                        00001F37 R      03
IPC$_ASTDEL                 =   00000002
IPL$_QUEUEAST               =   00000006
IPL$_SYNCH                  =   00000008
IPL$_TIMER                  =   00000008
IPL$_XE_DIPL                =   00000015
IPL$_XE_FIPL                =   00000008
IRP$B_EFN                   =   00000022
IRP$B_PRI                   =   00000023
IRP$B_RMOD                  =   0000000B
IRP$B_TYPE                  =   0000000A
IRP$B_XE_DATAP                  00000046  G
IRP$B_XE_FUNC                   00000021  G
IRP$B_XE_RING                   00000039  G
IRP$B_XE_SLOT                   00000038  G
IRP$C_LENGTH                =   000000C4
IRP$C_XE_STD                    00000060  G
IRP$L_AST                   =   00000010
IRP$L_ASTPRM                =   00000014
IRP$L_BCNT                  =   00000032
IRP$L_DIAGBUF               =   0000004C
IRP$L_FQFL                  =   00000060
IRP$L_IOSB                  =   00000024
IRP$L_IOST1                 =   00000038
IRP$L_IOST2                 =   0000003C
IRP$L_LBOFF                 =   00000090
IRP$L_MEDIA                 =   00000038
IRP$L_PID                   =   0000000C
IRP$L_RBOFF                 =   00000098
IRP$L_RBUFH_AD              =   00000094
IRP$L_SVAPTE                =   0000002C
IRP$L_UCB                   =   0000001C
IRP$L_WIND                  =   00000018
IRP$L_XE_DATBUF                 0000003C  G
IRP$L_XE_DGUNI                  00000040  G
IRP$L_XE_MAP                    0000003C  G
IRP$L_XE_P2BUF                  00000040  G
IRP$L_XE_SHR                    00000090  G
IRP$L_XE_SYSBUF                 0000003C  G
IRP$L_XE_UPADR                  00000044  G
IRP$L_XE_USERBUF                0000003C  G
IRP$M_CHAINED               =   00000020
IRP$M_COMPLX                =   00000008
IRP$M_DIAGBUF               =   00000080
IRP$Q_STATION               =   00000040
IRP$Q_XE_PCB                    00000094  G
IRP$V_CHAINED               =   00000005
IRP$V_DIAGBUF               =   00000007
IRP$V_FUNC                  =   00000001
IRP$W_BCNT                  =   00000032
IRP$W_BOFF                  =   00000030
IRP$W_CHAN                  =   00000028
IRP$W_FUNC                  =   00000020
IRP$W_SIZE                  =   00000008
IRP$W_STS                   =   0000002A
IRP$W_XE_CODE                   00000040  G
IRP$W_XE_P2SIZ                  00000044  G
IRP$W_XE_PROTYP                 0000003A  G
IRP$W_XE_RID                    00000038  G
IRP$W_XE_STATUS                 0000003A  G
IRP$W_XE_USERSIZ                00000038  G
IRP_ABORTED                     000024C9 R      03
JIB$L_BYTCNT                =   00000020
JIB$L_BYTLM                 =   00000024
LINE_CNT_BUFSIZ             =   0000005E
LINE_CNT_SIZE               =   00000011
LINE_CTR                        0000016D R      03
LINE_CTR_BUFSIZ             =   0000000C
LINE_CTR_SIZE               =   00000003
LINE_PARAM                      0000008F R      03
LINE_PARAM_WO                   00000086 R      03
LINE_PRM_BOFSIZ             =   000000BA
LOADMEM                         00000CCD R      03
LOAD_PORT                       000014BC RG     03
MASKA                       =   00000000
```

```
MASKL                           = 20000000
MATCH_ADDRESS                     000030A2 R      03
MATCH_MULTI                       00003073 R      03
MATCH_PROMTYP                     00003121 R      03
MATCH_PROTYP                      000030FF R      03
MATCH_SHR                         0000062D R      03
MATCH_SRC                         00001D77 R      03
MAX_C_CHAIN                     = 00000001
MAX_C_CTR                       = 00000020
MAX_C_MLT                       = 0000000A
MAX_C_RCV                       = 00000008
MAX_C_XMT                       = 00000004
MAX_PKT_SIZE                    = 000005DC
MIN_PKT_SIZE                    = 0000002E
MMG$GL_SPTBASE                    ********    X   03
MOPCTRTAB                         00000191 R      03
MOP_CTR_BUILD                     00002151 R      03
MOP_CTR_REQUEST                   00002110 RG     03
MOP_CTR_SIZE                    = 00000041
MOVE_MULTI                        00002B3F R      03
NEXTMSG                           000020A4 RG     03
NI_CTR_PROTYP                   = 00000260
NI_CTR_READ                     = 00000009
NI_CTR_REPLY                    = 0000000B
NMA$C_ACC_EXC                   = 00000003
NMA$C_ACC_LIM                   = 00000002
NMA$C_ACC_SHR                   = 00000001
NMA$C_CTCIR_BRC                 = 000003E8
NMA$C_CTCIR_BSN                 = 000003E9
NMA$C_CTCIR_DBR                 = 000003F2
NMA$C_CTCIR_DBS                 = 000003F3
NMA$C_CTCIR_MNE                 = 00000A8D
NMA$C_CTCIR_UBU                 = 00000429
NMA$C_CTLIN_BID                 = 000003F5
NMA$C_CTLIN_BRC                 = 000003E8
NMA$C_CTLIN_BS1                 = 000003F6
NMA$C_CTLIN_BSM                 = 000003F7
NMA$C_CTLIN_BSN                 = 000003E9
NMA$C_CTLIN_CDC                 = 00000425
NMA$C_CTLIN_DBR                 = 000003F2
NMA$C_CTLIN_DBS                 = 000003F3
NMA$C_CTLIN_LBE                 = 00000411
NMA$C_CTLIN_MBL                 = 000003F4
NMA$C_CTLIN_MBS                 = 00000A8D
NMA$C_CTLIN_MBY                 = 000003EA
NMA$C_CTLIN_MSN                 = 00000A8E
NMA$C_CTLIN_OVR                 = 00000428
NMA$C_CTLIN_RFL                 = 00000426
NMA$C_CTLIN_SBU                 = 00000429
NMA$C_CTLIN_SFL                 = 00000424
NMA$C_CTLIN_UBU                 = 0000042A
NMA$C_CTLIN_UFD                 = 00000427
NMA$C_CTLIN_ZER                 = 00000000
NMA$C_LINCN_LOO                 = 00000001
NMA$C_LINCN_NOR                 = 00000000
NMA$C_LINMC_CAL                 = 00000003
NMA$C_LINMC_CLR                 = 00000002
NMA$C_LINMC_SDF                 = 00000004
NMA$C_LINMC_SET                 = 00000001
NMA$C_LINPR_NI                  = 00000006
NMA$C_LINPR_POI                 = 00000000
NMA$C_PCCI_MST                  = 00000AFA
NMA$C_PCLI_ACC                  = 00000B1E
NMA$C_PCLI_BFN                  = 00000451
NMA$C_PCLI_BSZ                  = 00000B20
NMA$C_PCLI_BUS                  = 00000AF1
NMA$C_PCLI_CON                  = 00000456
NMA$C_PCLI_CRC                  = 00000B1C
NMA$C_PCLI_DCH                  = 00000B1B
NMA$C_PCLI_DES                  = 00000B21
NMA$C_PCLI_EKO                  = 00000B1F
NMA$C_PCLI_HBQ                  = 00000B1D
NMA$C_PCLI_HWA                  = 00000488
NMA$C_PCLI_ILP                  = 00000B17
NMA$C_PCLI_MCA                  = 00000B0F
NMA$C_PCLI_MLT                  = 00000B19
NMA$C_PCLI_PAD                  = 00000B1A
NMA$C_PCLI_PHA                  = 00000B04
NMA$C_PCLI_PRM                  = 00000B18
NMA$C_PCLI_PRO                  = 00000458
NMA$C_PCLI_PTY                  = 00000B0E
NMA$C_STATE_OFF                 = 00000001
NMA$C_STATE_ON                  = 00000000
NMA$M_CNT_COU                   = 00008000
NMA$M_CNT_MAP                   = 00001000
NMA$M_CNT_TYP                   = 00000FFF
NMA$V_CNT_MAP                   = 0000000C
NMA$V_CNT_WID                   = 0000000D
NO_SHR                            0000048E R      03
ORB$B_FLAGS                     = 0000000B
ORB$L_OWNER                     = 00000000
ORB$M_PROT_16                   = 00000001
ORB$W_PROT                      = 00000018
P1                              = 00000000
P2                              = 00000004
P2B_B_SPARE                       0000000B
P2B_B_TYPE                        0000000A
P2B_C_LENGTH                      0000000C
P2B_L_BUFFER                      00000004
P2B_L_POINTER                     00000000
P2B_T_DATA                        0000000C
P2B_W_SIZE                        00000008
P3                              = 00000008
P4                              = 0000000C
P5                              = 00000010
PBB_B_FUNC                        00000000  G
PBB_B_SUBFUNC                     00000001  G
PBB_C_LENGTH                      00000008  G
PBB_K_LENGTH                    = 00000008
PBB_W_PCBB2                       00000002  G
PBB_W_PCBB4                       00000004  G
PBB_W_PCBB6                       00000006  G
PB_FC_V_DUMPMEM                 = 00000010  G
PB_FC_V_LOADMEM                 = 00000011  G
```

```
PB_FC_V_LSUADDR         = 00000001  G
PB_FC_V_NOOP            = 00000000  G
PB_FC_V_RCCNTS          = 0000000B  G
PB_FC_V_RCSTATUS        = 0000000F  G
PB_FC_V_RDCNTS          = 0000000A  G
PB_FC_V_RDDEFAULT       = 00000002  G
PB_FC_V_RDMODE          = 0000000C  G
PB_FC_V_RDMULTI         = 00000006  G
PB_FC_V_RDPHYAD         = 00000004  G
PB_FC_V_RDRING          = 00000008  G
PB_FC_V_RDSERAD         = 00000014  G
PB_FC_V_RDSTATUS        = 0000000E  G
PB_FC_V_RDSYSID         = 00000012  G
PB_FC_V_WTMODE          = 0000000D  G
PB_FC_V_WTMULTI         = 00000007  G
PB_FC_V_WTPHYAD         = 00000005  G
PB_FC_V_WTRING          = 00000009  G
PB_FC_V_WTSERAD         = 00000015  G
PB_FC_V_WTSYSID         = 00000013  G
PB_MOD_M_DMNT           = 00000200  G
PB_MOD_M_DRDC           = 00002000  G
PB_MOD_M_DTCR           = 00000008  G
PB_MOD_M_ECT            = 00000800  G
PB_MOD_M_ENAL           = 00004000  G
PB_MOD_M_HDX            = 00000001  G
PB_MOD_M_INTL           = 00000040  G
PB_MOD_M_LOOP           = 00000004  G
PB_MOD_M_PROM           = 00008000  G
PB_MOD_M_TPAD           = 00001000  G
PB_MOD_V_DMNT           = 00000009  G
PB_MOD_V_DRDC           = 0000000D  G
PB_MOD_V_DTCR           = 00000003  G
PB_MOD_V_ECT            = 0000000B  G
PB_MOD_V_ENAL           = 0000000E  G
PB_MOD_V_HDX            = 00000000  G
PB_MOD_V_INTL           = 00000006  G
PB_MOD_V_LOOP           = 00000002  G
PB_MOD_V_PROM           = 0000000F  G
PB_MOD_V_TPAD           = 0000000C  G
PCB$L_JIB               = 00000080
PCB$L_PID               = 00000060
PCB$L_STS               = 00000024
PCB$Q_PRIV              = 00000084
PCB$V_SSRWAIT           = 0000000A
PCBB0                     00000004  G
PCBB2                     00000006  G
PCSR0                     00000000  G
PCSR1                     00000002  G
PCSR2                     00000004  G
PCSR3                     00000006  G
PEND                      000016B3  R     03
PEND1                     000016AF  R     03
POINT                   = 00000001
POKE_USER                 000030CA  R     03
PR$_IPL                 = 00000012
PRM_B_FLAG                00000002
PRM_FLG_M_CHECK         = 00000010
PRM_FLG_M_INVALID       = 00000004
PRM_FLG_M_MAX           = 00000002
PRM_FLG_M_MIN           = 00000001
PRM_FLG_V_CDB           = 00000003
PRM_FLG_V_INVALID       = 00000002
PRM_FLG_V_MAX           = 00000001
PRM_FLG_V_MIN           = 00000000
PRM_OFF_M_VALUE         = 000003FF
PRM_OFF_M_WIDTH         = 0000FC00
PRM_OFF_S_VALUE         = 0000000A
PRM_OFF_S_WIDTH         = 00000006
PRM_OFF_V_VALUE         = 00000000
PRM_OFF_V_WIDTH         = 0000000A
PRM_TYP_M_CODE          = 00000FFF
PRM_TYP_M_STRING        = 00001000
PRM_TYP_V_STRING        = 0000000C
PRM_W_OFF                 00000003
PRM_W_TYPE                00000000
PRV$V_PHY_IO            = 00000016
QUEPKT                    00000BF8  R     03
RCV_B_FLAGS               00000005  G
RCV_B_SEGBH               00000004  G
RCV_C_LENGTH              0000000A  G
RCV_FDT                   00000576  RG    03
RCV_FLG_M_CRC           = 00000008  G
RCV_FLG_M_ENP           = 00000001  G
RCV_FLG_M_ERRS          = 00000040  G
RCV_FLG_M_FRAM          = 00000020  G
RCV_FLG_M_OFLO          = 00000010  G
RCV_FLG_M_OWN           = 00000080  G
RCV_FLG_M_STP           = 00000002  G
RCV_FLG_V_CRC           = 00000003  G
RCV_FLG_V_ENP           = 00000000  G
RCV_FLG_V_ERRS          = 00000006  G
RCV_FLG_V_FRAM          = 00000005  G
RCV_FLG_V_OFLO          = 00000004  G
RCV_FLG_V_OWN           = 00000007  G
RCV_FLG_V_STP           = 00000001  G
RCV_K_LENGTH            = 00000050
RCV_LERR_M_BUFL         = 00008000  G
RCV_LERR_M_MLEN         = 00000FFF  G
RCV_LERR_M_NCHN         = 00002000  G
RCV_LERR_M_UBTO         = 00004000  G
RCV_LERR_S_MLEN         = 0000000C  G
RCV_LERR_V_BUFL         = 0000000F  G
RCV_LERR_V_MLEN         = 00000000  G
RCV_LERR_V_NCHN         = 0000000D  G
RCV_LERR_V_OVRN         = 0000000C  G
RCV_LERR_V_UBTO         = 0000000E  G
RCV_START                 000005E7  RG    03
RCV_W_LENERR              00000006  G
RCV_W_RID                 00000008  G
RCV_W_SEGBL               00000002  G
RCV_W_SLEN                00000000  G
READBD                    00000D19  R     03
READMEM                   00000CE0  R     03
READST                    00000D31  R     03
```

```
READ_CIRC_CTR                   00000C0B R      03
REG_DUMP                        000021F6 RG     03
REQ_COM                         000017EC R      03
REQ_COM_ERR                     000017EF R      03
RESTART_DELTA                 = 01C9C380
RESTART_ROUT                    00002235 RG     03
RES_MULTI                       00002B67 R      03
RETURN_IRP                      0000228E R      03
RETURN_MULTI                    0000303F R      03
RETURN_P2                       00002DD4 RG     03
RHDR_B_SPARE                    0000000B
RHDR_B_TYPE                     0000000A
RHDR_C_DATA                   = 0000000E
RHDR_C_LENGTH                   0000001A
RHDR_G_DEST                     0000000C
RHDR_G_SRC                      00000012
RHDR_L_BUFFER                   00000004
RHDR_L_DATA                     00000000
RHDR_T_DATA                     0000000C
RHDR_W_SIZE                     00000008
RHDR_W_TYPE                     00000018
SAV_MULTI                       00002B53 R      03
SCAN_LISTS                      00001B44 RG     03
SCH$GL_PCBVEC                   ********   X    03
SCH$IOLOCKW                     ********   X    03
SCH$IOUNLOCK                    ********   X    03
SCHED_FORK                      00001904 RG     03
SCHED_FORKC                     00001902 RG     03
SCS$GB_SYSTEMID                 ********   X    03
SENSEMODE_FDT                   00000B15 RG     03
SETMODE_FDT                     000006B8 RG     03
SET_DESAD                       00002FEB R      03
SET_MULTI                       00002F13 R      03
SET_MULTIN                      00000AF9 R      03
SET_PHYAD                       00002FE0 R      03
SHR_B_STS                       0000000B  G
SHR_B_TYPE                      0000000A  G
SHR_C_LENGTH                    0000002A  G
SHR_C_QUEUES                  = 00000002
SHR_DEF                         0000046D R      03
SHR_G_DEST                      00000012  G
SHR_L_PID                       0000000C  G
SHR_L_QBL                       00000004  G
SHR_L_QFL                       00000000  G
SHR_Q_QUEUES                    00000018  G
SHR_Q_RCVMSG                    00000018  G
SHR_Q_RCVREQ                    00000020  G
SHR_STS_M_INITED              = 00000001  G
SHR_STS_V_INITED              = 00000000  G
SHR_UCB                         00000987 RG     03
SHR_W_CHAN                      00000010  G
SHR_W_QUOTA                     00000028  G
SHR_W_SIZE                      00000008  G
SHUT                            00000F12 R      03
SHUTDOWN                        00002559 RG     03
SHUTDOWN_PROTYP                 0000253A RG     03
SHUTDOWN_UNA                    000023C8 RG     03
SIZ...                        = 00000006
SQUEEZ_MULTI                    00002FAD R      03
SS$_ABORT                     = 0000002C
SS$_ACCVIO                    = 0000000C
SS$_BADPARAM                  = 00000014
SS$_BUFFEROVF                 = 00000601
SS$_COMMHARD                  = 000020C4
SS$_CTRLERR                   = 00000054
SS$_DATACHECK                 = 0000005C
SS$_DATAOVERUN                = 00000838
SS$_DEVACTIVE                 = 000002C4
SS$_DEVALLOC                  = 00000840
SS$_DEVINACT                  = 000020D4
SS$_DEVOFFLINE                = 00000084
SS$_DEVREQERR                 = 00000334
SS$_DISCONNECT                = 0000204C
SS$_DUPUNIT                   = 000021C4
SS$_ENDOFFILE                 = 00000870
SS$_EXQUOTA                   = 0000001C
SS$_INSFMAPREG                = 00000344
SS$_INSFMEM                   = 00000124
SS$_IVBUFLEN                  = 0000034C
SS$_MEDOFL                    = 000001A4
SS$_NOPRIV                    = 00000024
SS$_NORMAL                    = 00000001
SS$_OPINCOMPL                 = 000002D4
SS$_POWERFAIL                 = 00000364
SS$_TIMEOUT                   = 0000022C
SS$_TOOMUCHDATA               = 0000029C
START                           00000F40 RG     03
STARTIO                         00000EB5 RG     03
STARTUP                         00000EF9 R      03
START_CONT                      000017C2 R      03
START_RECEIVE                   00001429 RG     03
START_TIMER                     000012D2 R      03
STEST                           00000D2A R      03
STOP                            00000F38 R      03
STOPPR                          00000D23 R      03
STOP_TQE                        0000231B R      03
TAKE_QUOTA                      000008F5 R      03
TIMEOUT                         0000232D RG     03
TIMOUT                          00002310 R      03
TOSAMSG                         0000207B R      03
TOSSMSG                         0000207E R      03
TQE$B_RQTYPE                  = 0000000B
TQE$B_TYPE                    = 0000000A
TQE$C_LENGTH                  = 00000030
TQE$C_SSREPT                  = 00000005
TQE$C_SSSNGL                  = 00000001
TQE$L_FPC                     = 0000000C
TQE$L_RQPID                   = 0000002C
TQE$M_REPEAT                  = 00000004
TQE$Q_DELTA                   = 00000020
TQE$W_SIZE                    = 00000008
TQE_C_DELTA                   = 00000002
TQE_DELTA                     = 01312D00
TQE_TIMER                       00002298 RG     03
```

```
UCB$B_DEVCLASS          =  00000040
UCB$B_DEVTYPE           =  00000041
UCB$B_DIPL              =  0000005E
UCB$B_FIPL              =  0000000B
UCB$B_TYPE              =  0000000A
UCB$B_XE_ACC               000000D4  G
UCB$B_XE_BFN               000000D5  G
UCB$B_XE_CDBPRM            000000DD  G
UCB$B_XE_CON               000000DE  G
UCB$B_XE_CRC               000000DD  G
UCB$B_XE_DCH               000000DA  G
UCB$B_XE_EKO               000000DF  G
UCB$B_XE_ILP               000000E0  G
UCB$B_XE_MLT               000000DC  G
UCB$B_XE_MLTTBL            000000E8  G
UCB$B_XE_MST               00000177  G
UCB$B_XE_MULTI             000000E7  G
UCB$B_XE_PAD               000000D9  G
UCB$B_XE_PRM               000000DB  G
UCB$B_XE_PRO               000000D8  G
UCB$B_XE_SETPRM            000000CC  G
UCB$B_XE_SHRPRM            000000D6  G
UCB$C_NI_LENGTH         =  00000098
UCB$C_XE_CDBPRM         =  00000004
UCB$C_XE_LENGTH            00000184  G
UCB$C_XE_QUEUES         =  00000004
UCB$C_XE_SETPRM         =  0000000A
UCB$C_XE_SHRPRM         =  0000000B
UCB$G_XE_DES               000000CC  G
UCB$G_XE_MLTTBL            00000125  G
UCB$G_XE_MULTI             000000E9  G
UCB$G_XE_PHA               000000E1  G
UCB$L_CPID              =  00000020
UCB$L_CRB               =  00000024
UCB$L_DDB               =  00000028
UCB$L_DEVCHAR           =  00000038
UCB$L_DEVDEPEND         =  00000044
UCB$L_LINK              =  00000030
UCB$L_NI_HWAPTR         =  00000090
UCB$L_NI_MLTPTR         =  00000094
UCB$L_ORB               =  0000001C
UCB$L_PID               =  0000002C
UCB$L_SVAPTE            =  00000078
UCB$L_XE_AST               000000C0  G
UCB$L_XE_CDB               00000180  G
UCB$L_XE_CPID              000000BC  G
UCB$L_XE_DEFUSR            000000C4  G
UCB$L_XE_FFI               00000178  G
UCB$L_XE_PID               000000B8  G
UCB$L_XE_RBLCTR            0000016D  G
UCB$L_XE_RBYCTR            00000171  G
UCB$L_XE_SBLCTR            00000165  G
UCB$L_XE_SBYCTR            00000169  G
UCB$L_XE_STIRP             0000017C  G
UCB$M_INT               =  00000002
UCB$M_ONLINE            =  00000010
UCB$M_POWER             =  00000020
UCB$M_TEMPLATE          =  00002000
UCB$M_TIM               =  00000001
UCB$M_XE_INITED         =  00000001  G
UCB$M_XE_INTERLOCK      =  00004000  G
UCB$M_XE_PROTYP         =  00000004  G
UCB$M_XE_RESTART        =  00008000  G
UCB$M_XE_RUN            =  00000010  G
UCB$M_XE_SHARE          =  00000008  G
UCB$M_XE_STACK          =  00000040  G
UCB$M_XE_START          =  00000020  G
UCB$Q_XE_IOQS              000000A0  G
UCB$Q_XE_QUEUES            00000098  G
UCB$Q_XE_RCVMSG            000000A0  G
UCB$Q_XE_RCVREQ            000000A8  G
UCB$Q_XE_SHARE             00000098  G
UCB$Q_XE_XMTREQ            000000B0  G
UCB$V_ONLINE            =  00000004
UCB$V_POWER             =  00000005
UCB$V_TIMOUT            =  00000006
UCB$V_XE_INITED         =  00000000  G
UCB$V_XE_INTERLOCK      =  0000000E  G
UCB$V_XE_PROTYP         =  00000002  G
UCB$V_XE_RESTART        =  0000000F  G
UCB$V_XE_RUN            =  00000004  G
UCB$V_XE_SHARE          =  00000003  G
UCB$V_XE_STACK          =  00000006  G
UCB$V_XE_START          =  00000005  G
UCB$W_BCNT              =  0000007E
UCB$W_BOFF              =  0000007C
UCB$W_DEVBUFSIZ         =  00000042
UCB$W_DEVSTS            =  00000068
UCB$W_ERRCNT            =  00000082
UCB$W_REFC              =  0000005C
UCB$W_STS               =  00000064
UCB$W_UNIT              =  00000054
UCB$W_XE_BSZ               000000D6  G
UCB$W_XE_CTR               00000161  G
UCB$W_XE_HBQ               000000D2  G
UCB$W_XE_MNECTR            00000161  G
UCB$W_XE_PROTYP            000000CA  G
UCB$W_XE_QUOTA             000000C8  G
UCB$W_XE_TOTQUO            00000175  G
UCB$W_XE_UBUCTR            00000163  G
UDB_BUF_B_RDRBH            00000008  G
UDB_BUF_B_RELEN            00000009  G
UDB_BUF_B_TDRBH            00000002  G
UDB_BUF_B_TELEN            00000003  G
UDB_BUF_C_LENGTH           0000000C  G
UDB_BUF_K_LENGTH        =  0000000C
UDB_BUF_W_RDRBL            00000006  G
UDB_BUF_W_RRLEN            0000000A  G
UDB_BUF_W_TDRBL            00000000  G
UDB_BUF_W_TRLEN            00000004  G
UDB_CTR_C_LENGTH           00000002  G
UDB_CTR_K_LENGTH        =  00000040
UDB_CTR_L_ENTRY            00000000  G
UDB_MLT_C_LENGTH           00000006  G
```

```
UDB_MLT_G_ENTRY          00000000  G
UDB_MLT_K_LENGTH       = 0000003C
UDB_SID_C_LENGTH       = 0000003C
UNIT_INIT                00000205 RG     03
UPADDR                   00000CF3 R      03
VA$M_BYTE              = 000001FF
VA$S_VPN               = 00000015
VA$V_VPN               = 00000009
VALIDATE_P2              00002B7B RG     03
VALID_MULTI              00002E9F R      03
VALID_PHYAD              00002EDB R      03
VEC$B_DATAPATH         = 00000013
VEC$B_NUMREG           = 00000012
VEC$L_IDB              = 00000008
VEC$L_INITIAL          = 0000000C
VEC$L_START            = 0000001C
VEC$L_UNITINIT         = 00000018
VEC$W_MAPREG           = 00000010
WRITEID                  00000D03 R      03
XBUF_C_HEADER            0000000E
XBUF_G_DEST              00000000
XBUF_G_SRC               00000006
XBUF_L_UCB               00000008
XBUF_T_DATA              0000000E
XBUF_W_SIZE              0000000E
XBUF_W_TYPE              0000000C
XBUF_W_UNUSED            00000006
XE$C_DG_ESTEST         = 00000009  G
XE$C_DG_HALT           = 0000000A  G
XE$C_DG_LOADMEM        = 00000001  G
XE$C_DG_READID         = 00000005  G
XE$C_DG_READMEM        = 00000002  G
XE$C_DG_READST         = 00000008  G
XE$C_DG_STEST          = 00000007  G
XE$C_DG_STOP           = 00000006  G
XE$C_DG_UPADDR         = 00000003  G
XE$C_DG_WRITEID        = 00000004  G
XE$DDT                   00000000 RG     03
XE_CMD_V_BOOT          = 00000005  G
XE_CMD_V_ESTEST        = 0000000B  G
XE_CMD_V_GETCMD        = 00000002  G
XE_CMD_V_GETPCBB       = 00000001  G
XE_CMD_V_HALT          = 0000000E  G
XE_CMD_V_NOOP          = 00000000  G
XE_CMD_V_PDMD          = 00000008  G
XE_CMD_V_START         = 00000004  G
XE_CMD_V_STEST         = 00000003  G
XE_CMD_V_STOP          = 0000000F  G
XE_C_CNTSIZ            = 00000002
XE_C_CRC               = 00000004
XE_C_HEADER            = 0000000E
XE_C_STPRO             = 00000660
XE_DEVID_V_DELUA       = 00000001  G
XE_DEVID_V_DEUNA       = 00000000  G
XE_END                   0000312E RG     03
XE_FC_V_CANCEL         = 00000016  G
XE_FC_V_CHMODE         = 00000018  G
XE_FC_V_ESTEST         = 0000001A  G
XE_FC_V_HALT           = 0000001B  G
XE_FC_V_INIT           = 00000000  G
XE_FC_V_LDMEM          = 0000000D  G
XE_FC_V_LDRING         = 00000002  G
XE_FC_V_LDSID          = 00000010  G
XE_FC_V_MOPCTR         = 00000019  G
XE_FC_V_MULTI          = 0000000C  G
XE_FC_V_RCCNTS         = 0000000B  G
XE_FC_V_RDCNTS         = 0000000A  G
XE_FC_V_RDDPA          = 00000015  G
XE_FC_V_RDMEM          = 0000000E  G
XE_FC_V_RDSID          = 00000011  G
XE_FC_V_READST         = 00000014  G
XE_FC_V_RECV           = 00000008  G
XE_FC_V_RESTART        = 00000017  G
XE_FC_V_START          = 00000006  G
XE_FC_V_STEST          = 00000013  G
XE_FC_V_STOP           = 00000009  G
XE_FC_V_STOPPR         = 00000012  G
XE_FC_V_TESTD          = 00000001  G
XE_FC_V_UPST           = 0000000F  G
XE_FC_V_WTMODE         = 00000005  G
XE_FC_V_WTMULTI        = 00000004  G
XE_FC_V_WTPHYAD        = 00000003  G
XE_FC_V_XMIT           = 00000007  G
XE_FUNCTABLE             00000038 R      03
XE_PCSR0_M_COMAND      = 0000000F  G
XE_PCSR0_M_DNI         = 00000800  G
XE_PCSR0_M_FATL        = 00000200  G
XE_PCSR0_M_ILLINT      = 00000010  G
XE_PCSR0_M_INTE        = 00000040  G
XE_PCSR0_M_INTR        = 00000080  G
XE_PCSR0_M_PCEI        = 00004000  G
XE_PCSR0_M_RCBI        = 00000400  G
XE_PCSR0_M_RSET        = 00000020  G
XE_PCSR0_M_RXI         = 00002000  G
XE_PCSR0_M_SERI        = 00008000  G
XE_PCSR0_M_TXI         = 00001000  G
XE_PCSR0_M_USCI        = 00000100  G
XE_PCSR0_S_COMAND      = 00000004  G
XE_PCSR0_V_COMAND      = 00000000  G
XE_PCSR0_V_DNI         = 0000000B  G
XE_PCSR0_V_FATL        = 00000009  G
XE_PCSR0_V_ILLINT      = 00000004  G
XE_PCSR0_V_INTE        = 00000006  G
XE_PCSR0_V_INTR        = 00000007  G
XE_PCSR0_V_PCEI        = 0000000E  G
XE_PCSR0_V_RCBI        = 0000000A  G
XE_PCSR0_V_RSET        = 00000005  G
XE_PCSR0_V_RXI         = 0000000D  G
XE_PCSR0_V_SERI        = 0000000F  G
XE_PCSR0_V_TXI         = 0000000C  G
XE_PCSR0_V_USCI        = 00000008  G
XE_PCSR1_M_DEVID       = 00000070  G
XE_PCSR1_M_ICAB        = 00004000  G
XE_PCSR1_M_PCTO        = 00000080  G
```

```
XE_PCSR1_M_SELF_TEST            = 00003F00  G
XE_PCSR1_M_STATE                = 0000000F  G
XE_PCSR1_M_XPWR                 = 00008000  G
XE_PCSR1_S_DEVID                = 00000003  G
XE_PCSR1_S_PCTO                 = 00000001  G
XE_PCSR1_S_SELF_TEST            = 00000006  G
XE_PCSR1_S_STATE                = 00000004  G
XE_PCSR1_V_DEVID                = 00000004  G
XE_PCSR1_V_ICAB                 = 0000000E  G
XE_PCSR1_V_PCTO                 = 00000007  G
XE_PCSR1_V_SELF_TEST            = 00000008  G
XE_PCSR1_V_STATE                = 00000000  G
XE_PCSR1_V_XPWR                 = 0000000F  G
XE_STAT_V_NIHALT                = 00000006  G
XE_STAT_V_NIUHALT               = 00000007  G
XE_STAT_V_PHALT                 = 00000008  G
XE_STAT_V_PRIMLD                = 00000001  G
XE_STAT_V_READY                 = 00000002  G
XE_STAT_V_RESET                 = 00000000  G
XE_STAT_V_RUN                   = 00000003  G
XE_STAT_V_UHALT                 = 00000005  G
XM$M_STS_ACTIVE                 = 00000800
XM$V_ERR_FATAL                  = 00000010
XM$V_ERR_START                  = 00000017
XM$V_STS_ACTIVE                 = 0000000B
XM$V_STS_BUFFAIL                = 0000000C
XM$V_STS_DISC                   = 0000000E
XM$V_STS_TIMO                   = 00000009
XMT_ALT_START                     000004A8 RG    03
XMT_B_FLAGS                       00000005  G
XMT_B_SEGBH                       00000004  G
XMT_C_LENGTH                      0000000A  G
XMT_C_TIM                       = 00000008
XMT_FDT                           000002D6 RG    03
XMT_FFI_START                     00000389 RG    03
XMT_FLG_M_DEF                   = 00000004  G
XMT_FLG_M_ENP                   = 00000001  G
XMT_FLG_M_ERRS                  = 00000040  G
XMT_FLG_M_MORE                  = 00000010  G
XMT_FLG_M_MTCH                  = 00000020  G
XMT_FLG_M_ONE                   = 00000008  G
XMT_FLG_M_OWN                   = 00000080  G
XMT_FLG_M_STP                   = 00000002  G
XMT_FLG_V_DEF                   = 00000002  G
XMT_FLG_V_ENP                   = 00000000  G
XMT_FLG_V_ERRS                  = 00000006  G
XMT_FLG_V_MORE                  = 00000004  G
XMT_FLG_V_MTCH                  = 00000005  G
XMT_FLG_V_ONE                   = 00000003  G
XMT_FLG_V_OWN                   = 00000007  G
XMT_FLG_V_STP                   = 00000001  G
XMT_INITIATE                      000003FC RG    03
XMT_K_LENGTH                    = 00000028
XMT_START                         000003C1 RG    03
XMT_TERR_M_BUFL                 = 00008000  G
XMT_TERR_M_LCAR                 = 00000800  G
XMT_TERR_M_LCOL                 = 00001000  G
XMT_TERR_M_RTRY                 = 00000400  G
XMT_TERR_M_TDR                  = 000003FF  G
XMT_TERR_M_UBTO                 = 00004000  G
XMT_TERR_M_UFLO                 = 00002000  G
XMT_TERR_S_TDR                  = 0000000A  G
XMT_TERR_V_BUFL                 = 0000000F  G
XMT_TERR_V_LCAR                 = 0000000B  G
XMT_TERR_V_LCOL                 = 0000000C  G
XMT_TERR_V_RTRY                 = 0000000A  G
XMT_TERR_V_TDR                  = 00000000  G
XMT_TERR_V_UBTO                 = 0000000E  G
XMT_TERR_V_UFLO                 = 0000000D  G
XMT_TIM                         = 00000005
XMT_W_RID                         00000008  G
XMT_W_SEGBL                       00000002  G
XMT_W_SLEN                        00000000  G
XMT_W_TDRERR                      00000006  G
_$$_                            = 00000000
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	000002BA (698.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
$$$105_PROLOGUE	000000C5 (197.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
$$$115_DRIVER	0000312E (12590.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	38	00:00:00.06	00:00:01.48
Command processing	133	00:00:00.44	00:00:03.43
Pass 1	1705	00:00:57.39	00:03:27.35
Symbol table sort	0	00:00:05.19	00:00:15.86
Pass 2	588	00:00:16.19	00:00:55.35
Symbol table output	2	00:00:00.57	00:00:01.23
Psect synopsis output	3	00:00:00.02	00:00:00.13
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	2472	00:01:19.86	00:04:45.06

The working set limit was 3450 pages.
449570 bytes (879 pages) of virtual memory were used to buffer the intermediate code.
There were 260 pages of symbol table space allocated to hold 4340 non-local and 665 local symbols.
9502 source lines were read in Pass 1, producing 76 object records in Pass 2.
98 pages of virtual memory were used to define 89 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SHRLIB]NMALIBRY.MLB;1	1
_255DUA28:[SYS.OBJ]LIB.MLB;1	41
_255DUA28:[SYSLIB]STARLET.MLB;2	14
TOTALS (all libraries)	56

3975 GETS were required to define 56 macros.

There were no errors, warnings or information messages.

```
MACRO/LIS=LIS$:XEDRIVER/OBJ=OBJ$:XEDRIVER MSRC$:XEDRIVER/UPDATE=(ENH$:XEDRIVER)+EXECML$/LIB+SHRLIB$:NMALIBRY/LIB
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

XEDRIVER
LIS

XDDRIVER
LIS

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

XFDRIVER
LIS